

Microcomputer Interface

digital inputs - outputs codes

code	7	6	5	4	3	2	1	0
432	0	0	0	0	0	0	0	0
32	1	1	1	1	1	1	1	1
84	1	1	1	1	1	1	1	1
96	1	1	1	1	1	1	1	1
128	1	1	1	1	1	1	1	1
160	1	1	1	1	1	1	1	1
192	1	1	1	1	1	1	1	1
224	1	1	1	1	1	1	1	1

COMPUTER
INTERFACE
532.001

UNILAB[®]
Blackburn, England

input codes

C	D	range
1	2	0 to +10v
3	7	0 to +1v
5	6	0 to +100mv
8	9	0 to +1v
10	11	variable + only
13	14	-5 to +5v
16	17	-0.5 to +0.5v
20	21	-50 to +50mv
24	25	variable - & +
28	29	
30	31	

output switches
24v 1A max

analogue
output
0-2.55v

trigger
sensitivity

amplifier
gain

digital inputs or outputs

analogue
inputs
10v max

A
B
C
D

INSTRUCTION MANUAL

	CONTENTS	<i>Software Routine</i>	<i>Page No.</i>
SECTION 1	INTRODUCTION		3
SECTION 2	GETTING STARTED		4
.1	Connecting up		4
.2	A quick look round		4
.3	Interface inputs		5
.4	Interface outputs		5
.5	The first steps		6
SECTION 3	TIME, VELOCITY, ACCELERATION		8
.1	Simple timing	TIMER1	8
.2	Measuring velocity	VEL1	9
.3	Average velocity	VEL2	9
.4	Measuring acceleration	ACCEL1 ACCEL2	10
SECTION 4	ADVANCED TIMING APPLICATIONS		11
.1	Machine code timing	SHRTTIM LONGTIM	11
.2	Acceleration : 'g'	ACCEL3	13
.3	Frequency by counting	FREQCNT	13
.4	Frequency by period	PERIOD	14
SECTION 5	DATA INPUT AND OUTPUT		15
.1	Controlling the Interface		15
.2	Control and direction codes		16
.3	Delays		18
.4	The analogue input system		20
.5	Autorangeing	ANALOG2	22
.6	The Interface as data recorder	GENALOG 4INPUT	23
.7	The trigger	FLATOUT	24
.8	The analogue output		25
SECTION 6	CONTROL		27
.1	The relays		27
.2	White-line follower		28
.3	Relay applications	RLYOPT	30
SECTION 7	INFORMATION		31
.1	Specifications		31
.2	Electrical protection		31
.3	Sources of analogue signals		32
.4	Some external hardware		33
.5	The V.I.A.		34
.6	Bitwise comparison		34

		Page No.
SECTION 8	SOFTWARE DETAILS	36
.1	TIMER1	36
.2	VEL1	36
.3	VEL2	36
.4	ACCEL1	36
.5	ACCEL2	36
.6	SHRTTIM	36
.7	LONGTIM	36
.8	ACCEL3	36
.9	FREQCNT	37
.10	PERIOD	37
.11	ANALOG2	37
.12	GENALOG	37
.13	4INPUT	37
.14	FLATOUT	37
.15	RLYOPT	38
.16	ECG	38
.17	TRISCAN	38
.18	BALANCE	39

SECTION 1

INTRODUCTION

This book is intended to enable you to obtain the best from your Interface. In turn, the Interface will enable you to use your computer in a much wider range of applications. The Unilab Interface has been designed especially for use in scientific and control applications. It converts the computer into a laboratory tool with an extremely wide range of applications yet at relatively low cost. Some of the performance specifications are quite remarkable and the instrument will enable you to perform experiments which previously could have been effected only with very specialised equipment. A warning before you start - once you have mastered the basics of Interfacing to computers, the whole subject tends to be very addictive and absorbing!

Newcomers to the computer scene are often worried, understandably, that the business of Interfacing is going to raise a new set of problems when they are already concerned that their knowledge about the use of computers is small. However, the Interface and its companion computer can be treated as any other piece of laboratory instrumentation. It may be helpful to know how it works, but it is certainly not essential.

This book is written for beginners and experts. For the beginner, it is an instruction manual and for the expert a source of information which will enable him to create software for new applications.

Sample Programs

This book contains a number of sample programs. Some are very short and are included with the text. Longer programs are described at the end of the book from Page 36 onwards, and are available on the cassette, in sequence. Reference is made to them both by page number and Title. The Title referred to is that used on the cassette tape.

It is expected that the user will wish to 'mix and match' sections of programs in developing software for their own applications. Thus, the

user may wish to arrange to print listings from the cassette. Many have been written with explanatory REM statements to make the structure clear.

In addition, the user is strongly recommended to make a number of copies of the programs on cassette or disc to avoid accidental loss or corruption of the software.

Users up-grading to Disc-based Systems

The use of discs to store programs brings the advantage of speed in loading and saving, but with the penalty of reduced program space in memory. The software described above is suitable for use with BBC model B microcomputers fitted with the Acorn Disc Filing System. This takes the form of a cassette tape of re-written programs, one of which (GENALOG) has been 'sectionalised'. For example, the first section of 'GENALOG' is used to assemble a machine-code delay routine and then to load and run the second section which contains the main data-capture and analysis routines. The first program on the tape (named "T-TO-D") is one which allows the automatic transfer of the remainder of the tape to disc. We are grateful to "Acorn User" magazine for permission to include this useful routine. For those using cassette storage, all programs except "GENALOG" can be loaded and run in the usual way. A non-sectionalised version called "GNLGCSS" is included as the very last program on the tape.

New Software

From time to time, UNILAB will publish new software developed within the company, concentrating on programs of general applicability which employ the integrated nature of the interface's facilities as fully as possible.

UNILAB will develop a software clearing house through which interested users can exchange their ideas on software. Suitable programs submitted to us will be duplicated and made up into issues available at a nominal charge from UNILAB on a yearly basis.

SECTION 2

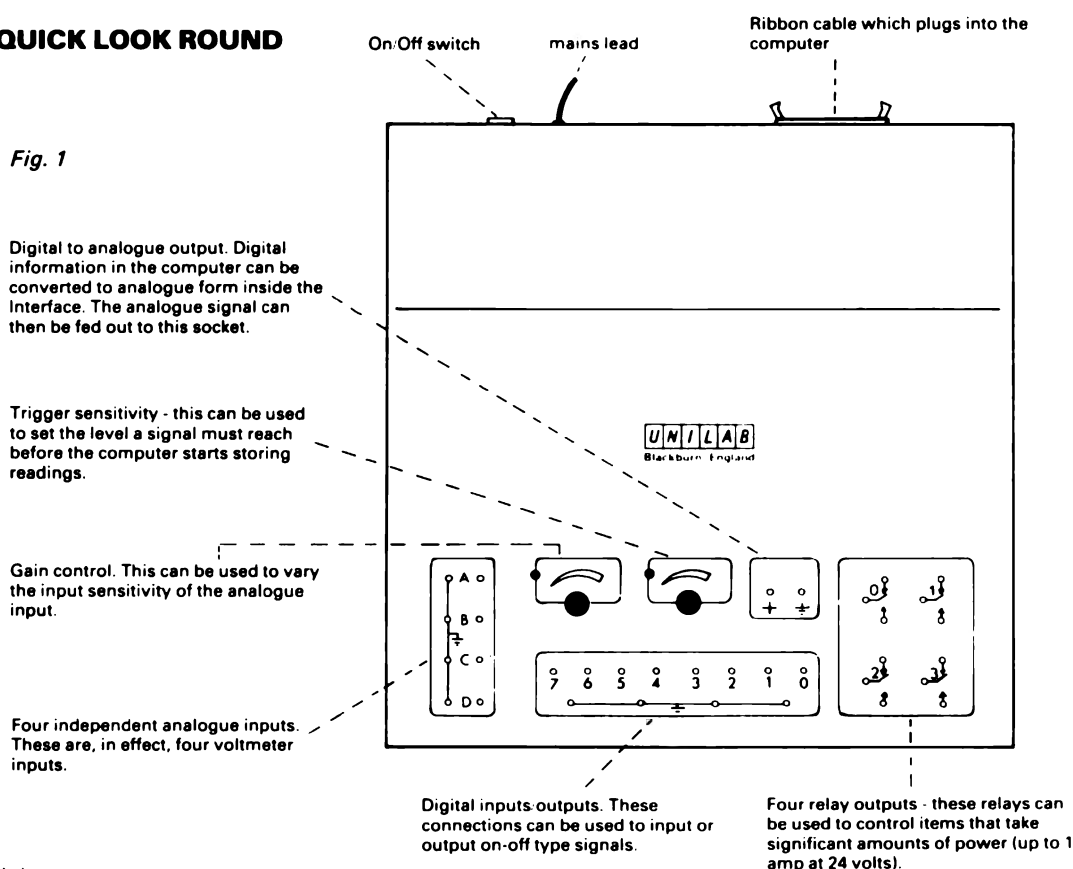
GETTING STARTED

2.1 CONNECTING UP

1. Switch on the mains supply to the computer and the Interface.
2. Note that the connectors at each end of the ribbon cable are *sockets* which are going to fit into *plugs* in the computer and the Interface!
3. Take one end of the ribbon cable and match the red edge of the cable to the 'arrow mark' of the '1 MHz BUS' plug underneath the BBC microcomputer. You will notice that the IDC header has a 'key' which permits the cable to be connected only one way round. Insert the IDC header onto the '1 MHz BUS' until the clips at each end lock onto the header. Although it doesn't matter electrically which end of the cable is used, you will find that one end is easier to use than the other.
4. Repeat the procedure to connect the cable to the Interface, again taking care to ensure that the 'key' on the IDC header matches the 'slot' in the plug on the Interface.
5. You are now ready to start! You will need to enter a program into the computer. In the following pages you will be taken through a number of exercises which illustrate the features of the Interface. The programs which are used to operate the machine can be entered by typing them in, or loaded from the accompanying cassette tape, using the program title shown.
6. Please note that *before loading any program* into the computer, the Interface should be connected to the computer and *should be switched on*. Strange effects can be produced if the Interface is connected but switched off; for example, corruption of the program, multiple letters from the keyboard, or even complete 'locking up' of the computer, are all possible. This is not a fault of the Interface but is a result of the way the computer responds to peripheral devices which can provide system 'interrupts'.

2.2 A QUICK LOOK ROUND

Fig. 1



2.3 INTERFACE INPUTS

Information from scientific experiments, which might be fed into the Interface input, can be divided into two broad groups, digital and analogue.

Digital information

This has only two valid states, ON and OFF. To represent these states in electronic equipment it is usual to use +5 volts to represent ON and 0 volts to represent OFF. Very often, ON and OFF are abbreviated to 'logic 1' and 'logic 0'. This information could be presented by a switch which might be mechanical or electronic. This switch might indicate when liquid had reached a certain level or it could be a thermostat indicating that a certain temperature had been achieved. However, the most usual use for such electronic switches in science experiments is in timing applications such as in velocity and acceleration experiments where a falling sphere or a moving trolley operates a microswitch or obstructs a light beam when it reaches a certain point on its journey.

Up to 8 digital inputs can be fed into the interface at one time. These inputs are standard TTL but with protection fitted. As with all TTL inputs, an unconnected input 'floats high', i.e., if nothing is connected to a particular input, it will assume a state of logic 1. This is a useful feature to have on input lines since it allows the user to apply a logic 1 without need for an external 5 volt power supply.

Analogue information

This can have any value between predetermined limits. For example, the P.D. across a photocell would be zero volts in total darkness but as the light intensity increased, the P.D. across the cell would rise smoothly up to some maximum. Computers cannot read analogue values directly but need an analogue to digital converter between the analogue voltage source and the digital data lines in the computer. This part of the Interface converts any incoming analogue voltage into a corresponding digital value which the computer can read and use.

2.4 INTERFACE OUTPUTS

Digital Outputs

Up to eight of the digital inputs/outputs can be programmed as outputs. In fact, these inputs/outputs are controlled in pairs so that one can have:-

No outputs	with	Eight inputs
or Two outputs	with	Six inputs
or Four outputs	with	Four inputs
or Six outputs	with	Two inputs
or Eight outputs	with	No inputs

The setting of these digital lines to inputs or outputs is achieved by means of the Control Codes, the use of which will be covered later in Section 5.2.

As they stand, the digital outputs provide only sufficient drive for TTL or CMOS type inputs. For higher power applications such as in the control of motors and lamps, four relays have been provided. In some instances, the relay operations are too slow and one needs to control power

directly from the digital lines. An example of such a case is in the use of the Interface to drive a stepper motor. The solution here is to use driver transistors or power FETs on each of the required digital outputs. Suitable circuits are given in Section 7.4.

The Relay Outputs

These four relays provide single pole changeover contacts capable of switching a current of up to 1 amp from a supply of up to 24 volts. When brought into operation by use of the appropriate Control Code, these relays are switched according to the state of the digital outputs 0, 1, 2 and 3. The relay sockets on the front panel are marked with the number of the digital line which controls them.

An important point about these relays is that the data can be latched into them. If they were not latched, they would continually reflect the state of the digital lines 0 to 3 and would be constantly changing whenever the Interface was running on inputs or outputs. The latch enables the user to set the state of the relays then use the Interface to

do other tasks without relays changing. Thus, for example, the relays may control the motors on a model car. After setting then latching the relays, the Interface could use its inputs to read off information from sensors on the car. As a result of these readings, the computer would decide what action was needed, and the relay latch would be lifted so that the relays could be changed as necessary.

The Analogue Output

The Interface contains a digital to analogue converter. This is used to convert digital information from the computer into an analogue voltage or current which can be fed to external equipment. The output can be fed to a recording instrument such as a chart recorder. Therefore information, acquired at very high speed through the analogue input into the computer for storage,

can be replayed out of the analogue output sufficiently slowly for a standard chart recorder.

The analogue output is latched so can be used to apply a P.D. to a circuit which can be held while the Interface changes to analogue input to read the effect of that applied P.D. The analogue output can then be changed and the whole process repeated. In this manner the response curve of a tuned circuit can be plotted by using the analogue output to sweep a signal generator over a range of frequencies. The same technique can be used to apply varying inputs to semiconductor devices. The output signals can then be plotted against inputs to obtain characteristic curves. Waveforms of greatly varying complexity can be created by storing appropriate digital values in a section of the computer memory then replaying these values continuously through the analogue output.

2.5 THE FIRST STEPS

(1) – using a digital input

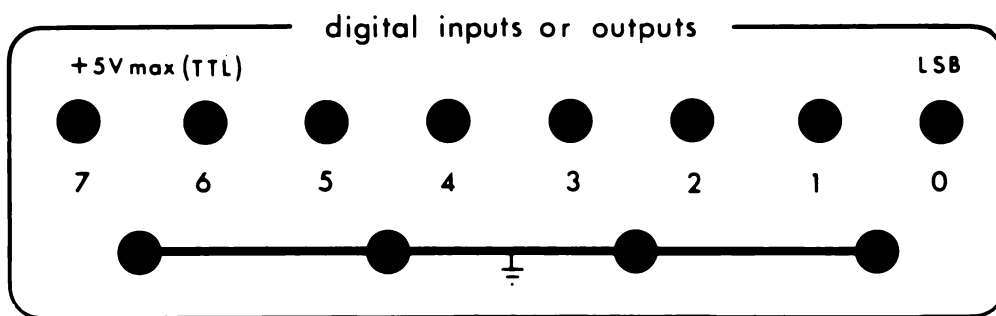


Fig. 2

The digital inputs are used to detect ON/OFF situations. You will have noticed that the connections are marked as inputs/outputs. We will consider their output role later.

Enter the program into the computer:

```

10 ?&FCC2=0:?&FCC3=255:?&FCC1=32
30 A=?&FCC0: B=A AND 1
50 IF B= 0 THEN VDU12:PRINT
   "Connected to ground. i.e., 0 volts"
60 IF B=1 THEN VDU12:PRINT
   "Open circuit. i.e. +5 volts"
70 GOTO 30

```

Having entered the program, you can try it out. Connect a short 4 mm lead between digital input '0' and the adjacent green earth socket. When you run the program it should tell you whether input '0' is earthed or left open circuit.

An important point to note here is that leaving a digital input unconnected allows it to 'float high', i.e., leaving a digital input unconnected has the same result as a +5 volt input.

(2) – connecting a photodetector

If you have a photodetector as is used in air track or trolley experiments (e.g., Unilab 414.026) you can connect this between input '0' and earth in place of the 4 mm lead (with 414.026 the larger terminal plugs into the earth socket).

If the program is now run as before, you will see that the computer can now sense when light entering the photodetector is shut off. Whilst this may seem an expensive way to detect the presence of light, it is the basis on which the Interface and computer can be used as a very sophisticated timing machine for use in velocity

and acceleration experiments. You might like to tidy up this program when applied to the light detector by changing lines 50 and 60 to:

```
50 IF B=0 THEN VDU12:PRINT "Light"
```

```
60 IF B=1 THEN VDU12:PRINT "Dark"
```

NOTE: Line 30 in this program contains the instruction which picks out information on digital input 0. For a fuller explanation of how to 'pick out' other lines, or combinations of lines see Section 7.6 on 'Bitwise Comparison'.

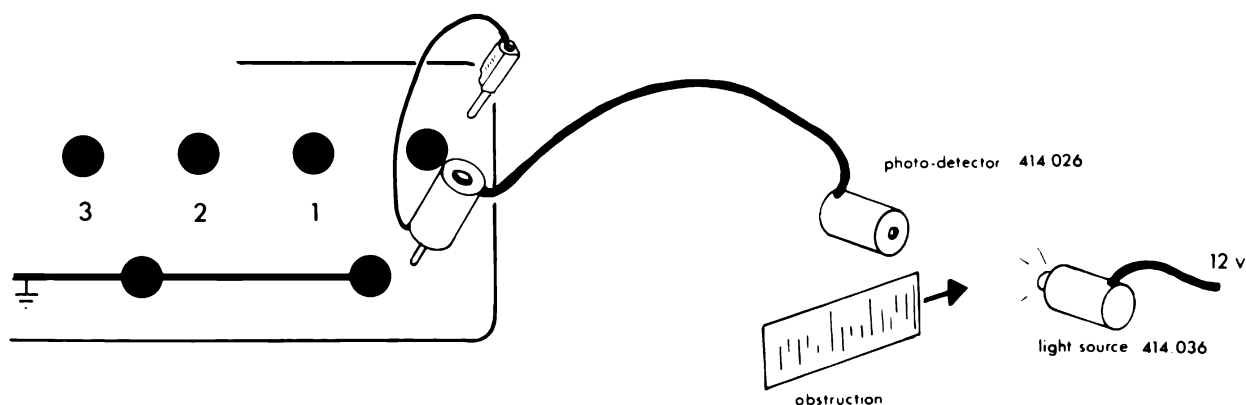


Fig. 3

SECTION 3

TIME, VELOCITY, ACCELERATION

3.1 SIMPLE TIMING

The computer contains a clock that counts in one hundredths of a second. To read the time, one simply enters PRINT TIME (return)*

To set the clock to zero, type in TIME = 0 (return).

To use this in an example, suppose we wish to time the interval between input 0 going open circuit (logic 1) and being connected to earth again (logic 0). Exactly the same program would time the period for which the light beam to a photodetector was interrupted. The stages are as follows:

- Wait for input 0 to go from logic 0 to logic 1 (0 volts to 5 volts).
- Read the time and put its value into a variable FIRSTTIME.
- Wait for input 0 to return to '0' (0 volts).
- Read the time and put its value into a variable LASTTIME.
- Subtract FIRSTTIME from LASTTIME to obtain the period elapsed in hundredths of a second.
- Divide by one hundred to convert it to seconds.
- Print the result.

* (return) means 'press the return key' as distinct from RETURN which would mean 'type in the word RETURN'.

Enter the program into the computer: Page 36 "TIMER1"

Once loaded into the computer, the program can be tried out. First of all use a 4 mm lead connected between input 0 and the adjacent green ground socket. When this lead is disconnected from input 0, the input will 'float high' and timing will start. On reconnecting to ground, the timing will stop and the total elapsed time will be displayed.

The lead can then be replaced with a photodetector with associated lamp (e.g., Unilab 414.026 Phototransistor with 414.031 Light Source). [See Fig. 3]. When run, the program will then time the period for which the light beam is obstructed.

Timing while an input is low

In the last example, the unit started timing when input 0 went high (5 volts) i.e., when the P.D. on input 0 rose from 0 to +5 volts. To reverse this so that timing starts when input 0 goes low, it is merely necessary to change lines 160 and 260:

```
160 UNTIL B=0
260 UNTIL B=1
```

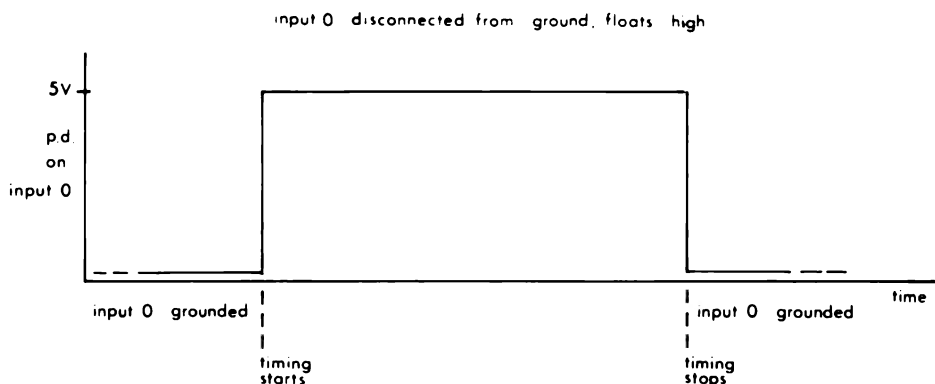


Fig. 4

3.2 MEASURING VELOCITY

The simple timing program can, of course, be used to measure velocity if the moving object carries a light beam obstructor of known length.

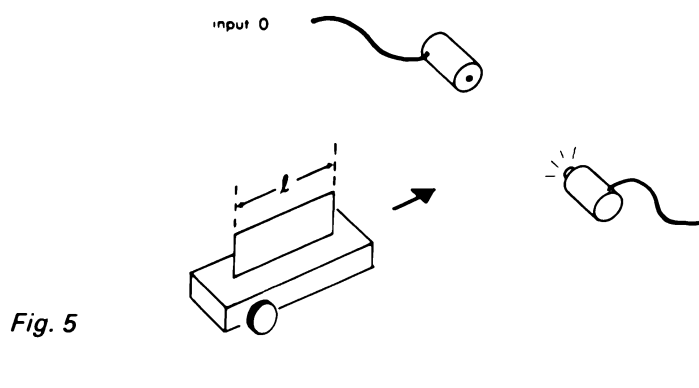
To compute velocity, the computer needs to know the length of this card, so only a few lines must be added to our simple timing program. This has already been done in the next program.

Enter the program into the computer - Page 36
"VEL1"

Important Note

In this and subsequent timing programs, the computer calculates the final results to a far higher precision than is justified either by the method or by the resolution of the clock. The programs have normally been adapted to express the results in 'rounded up' form to give a reasonable number of decimal places,

e.g., `500 PRINT (INT(100*RESULT + 0.5))/100`
would express RESULT to 2 decimal places.



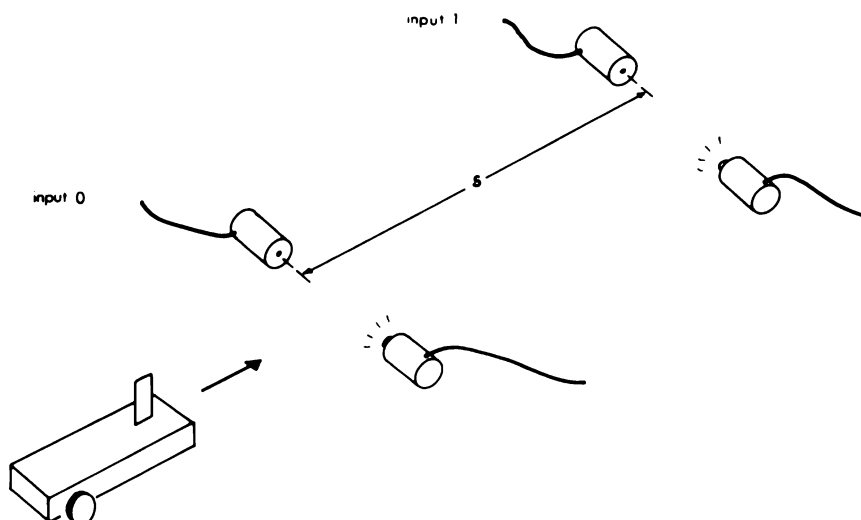
3.3 AVERAGE VELOCITY

To obtain greater precision, it is common to use two light gates to measure the average velocity of a moving object. The same apparatus can be used to measure 'g' by free fall.

In this case, the time is noted when the first gate is operated and then noted again as the second gate is operated. From the distance between the gates, the average velocity can be computed.

Using the next program, the computer will start timing when the object passes the light gate connected to input 0. Timing stops when the light gate connected to input 1 is interrupted. The user is then asked to give the distance between the gates and the computer then calculates the average velocity and displays it on the screen together with the time and distance information.

Enter the program into the computer - Page 36
"VEL2"



3.4 MEASURING ACCELERATION

There are two approaches, using two light gates cut by a single obstructor of known length, or using a single light gate cut by a double obstructor of known dimensions.

Approach A

As the trolley passes the first photodetector, the time for which the beam is interrupted is measured. This time is t_1 .

Some time later, the second photodetector beam is interrupted for a time t_2 .

Therefore the initial velocity, $u = \frac{l}{t_1}$

and the final velocity, $v = \frac{l}{t_2}$

These velocities are actual trolley velocities half way through the time intervals t_1 and t_2 . The computer also measures and computes the time between mid-point times; this time is known as 'interval'.

The acceleration is then calculated using

$$a = \frac{v - u}{\text{interval}}$$

The next program, which is a modified version of the former examples, will perform this task.

The first and last photoelectric or mechanical gates are connected to digital inputs 0 and 1 on the Interface.

Enter the program into the computer Page 36 "ACCEL1"

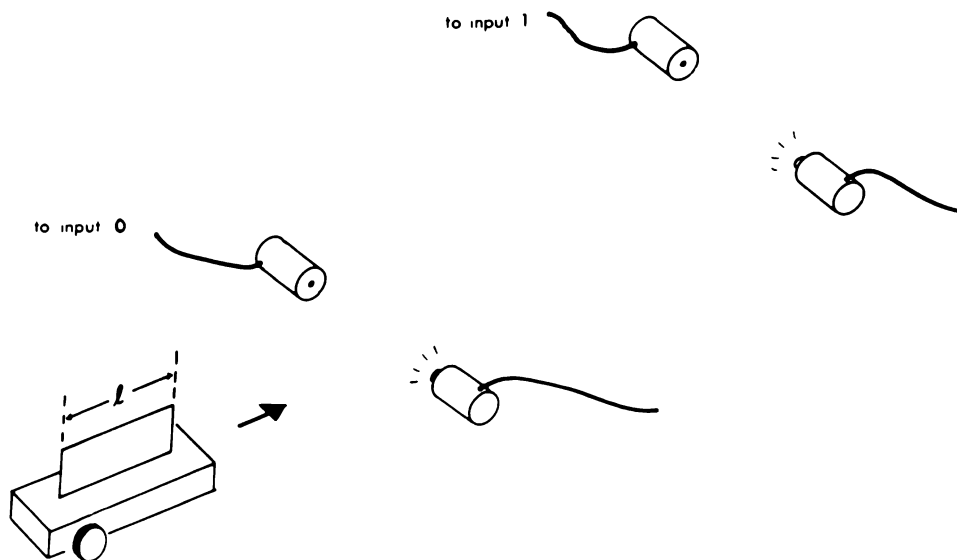


Fig. 7

Approach B - This second approach uses the same basic principle but requires the use of only a single photoelectric gate.

In this case, two 'light beam interrupters' are mounted on the moving object and are set at known distances apart. In practice, it is convenient to make the 'interrupters' from the same piece of card. Both lengths ' l ' must be the same.

Enter the program into the computer Page 36 "ACCEL2"

NOTE: Up to this point, all programs have used the centi-second clock TIME of the computer. The accuracy of time measurements, and calculations based upon them, will be very poor where the obstructors are small and/or the velocities are large. Good results for 'g by free fall' cannot be expected.

SECTION 4

ADVANCED TIMING APPLICATIONS

4.1 MACHINE CODE TIMING

Operations in 'Basic' are executed relatively slowly by a microcomputer. Whilst in the previous programs, the degree of precision may be sufficient for most purposes, timing exercises in 'Basic' do not make fullest use of a computer's ability to time events to great precision.

The Unilab Interface contains an Input/Output (I/O) chip with timers. These timers can time in intervals of 1 microsecond with the precision of the quartz clock crystal inside the microcomputer.

To take advantage of the high counting rate of these clocks, the timing part of the program needs to be executed in machine code. The simplest way to enter this code is to use the Mnemonic Assembler in the microcomputer. If you have not used an Assembler before, the thought may seem rather daunting. For this reason, we have provided a number of routines which the user can copy and use as needed without necessarily understanding how they work. The more advanced user may wish to turn to Page 34 which provides factual information about the input/output circuitry in the Interface.

There is one important point that has to be remembered when using software timing. To detect a change of an input, the software runs the microprocessor round a loop of instructions. This loop takes a finite amount of time to execute, typically between 7 and 50 microseconds depending on its length. Within this loop there is only one point when the microprocessor 'reads'

the input to see if it has changed. Thus there may be some delay in the timer starting. The size of this delay will vary according to the point the microprocessor has reached in the loop when the input changes. Similarly there will be a range of possible delays which may be incurred through the loop at the end of the timing period. Whilst for most of the timing period, the time is measured at high resolution and accuracy, the loops at each end of the timing routine produce an error which can be significant if the timing period is extremely short. Figure 8 shows this diagrammatically.

These errors are minimised if the time taken to circulate each program loop is kept as short as possible. These loops can be very short provided the microprocessor does not need to look for the 'timing out' of a timer. The timer inside the Interface will count to a maximum of 65536 microseconds, i.e., approximately 65 milliseconds. By using a 'time out flag', this maximum measurable period can be doubled. If periods greater than about 130 milliseconds are to be timed, one needs a more complex program with longer loops. Such a program will note how many times the timer timed-out during the 'on' period and will take this into account when calculating the total elapsed time.

Two forms of machine code timing program are given. The first will time periods of up to 131072 microseconds with an error of ± 15 microseconds. The second program will time periods of up to rather more than 4000 seconds but the error is ± 60 microseconds.

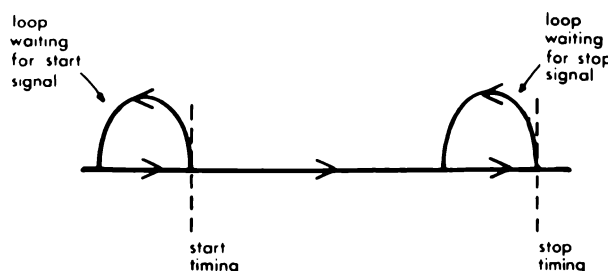


Fig. 8

Machine code timing program – for periods of up to 130 milliseconds

Enter the program into the computer - Page 36
"SHRTTIM"

- (a) The timing period must not exceed 131070 microseconds. If it does, the computed elapsed time will be incorrect.
- (b) Timings can be made on any of the eight inputs 0 - 7.
- (c) Timing can be started and stopped by one input or can be started by one input and stopped by another.
- (d) Only one period can be measured at a time, but a number of periods can be measured in rapid succession.
- (e) Timing is started and stopped by inputs going from logic 0 to 1 or vice versa. Remember that an illuminated phototransistor produces a logic 0.
- (f) The sequence of operations is as follows:
 - (i) Enter the program into the computer.
 - (ii) When run, the program will ask which input is to be used to start the timing. Enter the number of the input you wish to use.
 - (iii) You will then be asked which input is to be used for stopping the timing. If the same input is being used for start and stop, then the number entered will be the same as in (ii) above.
 - (iv) You are then asked if the start is to be a high or low going signal
 - (v) The same question is asked in regard to the stop signal.
 - (vi) When the above procedure has been completed the timing program is ready and when a computer key is pressed, timing can start when the 'start' input is activated.
 - (vii) The time elapsed is stored in a variable called 'MICROSEC'. The result can therefore be used in your own parts of the program by calling this variable. You might, for example, store the results from a series of experiments in an array.
 - (viii) To run another sample, there is no need to go through the whole procedure detailed above. Pressing 'S' (for same) will take the machine directly to the start of the timing operation. If you wish to change the conditions of the experiment, enter 'C'.

Sample run:

The program has been loaded. The Interface is to be used to time the period for which a passing trolley obstructs a light gate. The phototransistor is connected to input 0 on the Interface.

RUN

Which input is being used to start the timing?

(Answer) 0

Which input is being used to stop the timing?

(Answer) 0

Do you want timing to start on a high or a low input level? (H/L)

(Answer) H

Do you want timing to stop on a high or a low input level? (H/L)

(Answer) L

Press any key when the apparatus is ready for timing to start

(Answer) (Press any key then release the trolley)

.

.

Time elapsed was 58676 microseconds.

Press 'S' for a repeat with the SAME experimental conditions.

Press 'C' if you wish to repeat but with a CHANGE to experimental conditions.

Machine code timing program for periods of up to 4250 seconds

The advantage of this program over the previous one is that it will display times of up to 4250 seconds measured in microseconds. The disadvantage is that the time loop times are greater than in the last program and the possible error is ± 60 microseconds. It should be mentioned that the accuracy is also affected by the accuracy of the quartz crystal inside the computer itself although as a proportion of the total reading this will normally be insignificant. The program is used in much the same way as the last one except that input 6 is used by the program so can not be used for an input signal.

Enter the program into the computer - Page 36
"LONGTIM"

4.2 ACCELERATION : 'g'

On Page 10 a method for measuring acceleration with a single photoelectric gate was described. That program was written in "Basic" which may prove too slow in some applications. By using a modified form of the "LONGTIMER" program, the same experiment may be performed with greater resolution and accuracy.

A double light beam obstructor (see Fig 9) is used as before. When the program is run, the computer will ask for information about dimensions and will then be ready to make the required measurements.

Enter the program into the computer - Page 36
"ACCEL3"

This program is able to handle short time periods and/or high velocities, so is suitable for the measurement of the acceleration due to gravity, 'g'.

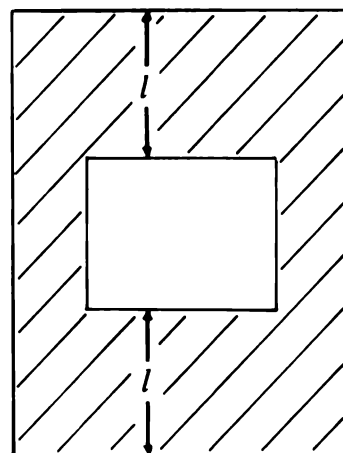


Fig. 9

NOTE: The transitions from logic 0 to logic 1, and back to logic 0, must be extremely sharp in this application. If this is not the case, it may happen that the program sees the first transition as *all* transitions needed for it to produce the answer. See page 33 for a method of sharpening the output from a phototransistor.

4.3 FREQUENCY BY COUNTING

The frequency of a source can be measured in two main ways:

- (a) By counting the number of complete cycles produced by the source in a known length of time.
- (b) By measuring the time taken for the source to produce one complete cycle. This is often referred to as 'period mode' and is very useful when great precision is required on a low frequency source.

To obtain reliable counting the Interface must receive a square wave input that switches between approximately 0 and +5 volts. If the signal source is not producing a square wave, it will be necessary to place a Schmitt trigger between the output of the source and the input to the Interface. (see Section 7.4).

Frequency measurement by counting

The Interface contains a device which can count up to 65536 pulses which are applied to digital input 6. The internal timer used before can be used to fix the period for which the count

continues, up to a maximum of 65536 microseconds. Ideally, the sampling time should be set to as long a period as is possible without the counter over-ranging. The ability of the computer to examine data and subsequently make decisions make it possible to render the frequency counter 'auto-ranging'.

The sample program sets the Interface up to count initially over the maximum period of 65536 microseconds. After the count is complete, a flag is checked to see if the count has gone over-range. If not, the frequency is computed and displayed but if an over-range has occurred, the sampling period is reduced and another sample taken. This process is repeated until a period is found which will accommodate the count.

Enter the program into the computer - Page 37
"FREQCNT"

NOTE: This program is particularly suitable for frequencies greater than 2000 Hz if accuracies better than $\pm 1\%$ are required.

4.4 FREQUENCY BY PERIOD

This program is similar to the machine code program used to time periods up to 4250 seconds and is particularly useful when the frequency of the source is very low. The opportunity has been taken to incorporate some advanced features into this program in order that its scope can be widened.

So far we have timed and counted only digital type signals. In the practical laboratory, however, the input signals are not always clean square waves and there may be considerable noise on both the low and high portions of the signal. This noise may be sufficiently strong to cause false switching and loss of accuracy as a result.

A technique that can be used to overcome this noise problem is to set up a "window" which the signal must cross before an active transition is accepted.

To count as a "high" signal, the level must exceed the window-high level and will be accepted as a continuing "high" until it falls to a level below the window-low level. Once it has crossed this window-low level, the signal is accepted as a "low" and will continue as such until it rises above the window-high level again. (Fig. 10).

The program "PERIOD" uses the analogue input 'A' for signal input and starts by asking the user to input a sample signal. This is plotted on the screen and the window levels, superimposed on the plot, are adjusted as required. Once the user has selected the number of samples required, the timing can begin and the program will time successive "high" and "low" periods. These are subsequently displayed then placed into 40 bins of selectable width. The bin contents are finally displayed with a printer option.

Page 37 "PERIOD"

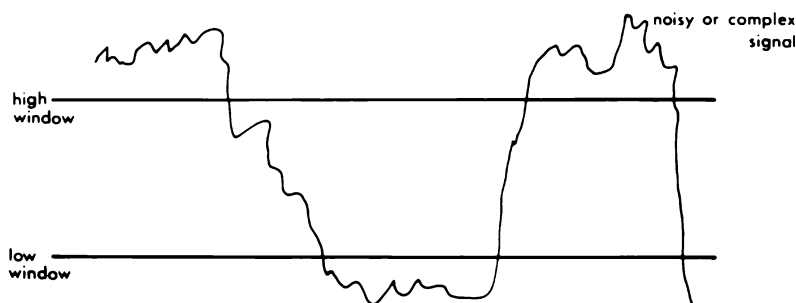


Fig. 10

SECTION 5

DATA INPUT AND OUTPUT

The Unilab Interface can operate in a number of different ways which fall broadly into two groups, input (read) operations and output (write) operations. The manner in which the Interface is operating is governed by the Control and Direction Codes of which more later. Whatever Interface function is in operation, the data is always routed through the same address.

The address for Data is &FCC0 (Hexadecimal FCC0)

If the computer is reading data from the Interface in an input mode, the operation it will be conducting could be:

$X = ?\&FCC0$ in 'Basic' where X can be any variable

or $PRINT ?\&FCC0$ in 'Basic' when print up of result is needed

or $LDA \&FCC0$ in 'Assembler'

To write data to the Interface for subsequent output, the operation could be:

$?\&FCC0 = X$ in 'Basic' where X is or contains the value for output

or $LDA \# 27 : STA \&FCC0$ in 'Assembler' when a value of 27 is to be output.

The above are just examples of the type of operation that are commonly found. Once the data has been obtained from the FCC0 address, the computer can manipulate it in any of the ways it might treat any data.

5.1 CONTROLLING THE INTERFACE

Before the Interface can be used, it has to be told by a computer how it is to operate. This operation is achieved in three stages which are SET UP, SET CONTROL CODE, SET DIRECTION CODE.

Set up

The Control Code is passed to the Interface by means of the contents of the address &FCC1. Initially, the VIA inside the Interface is told that the contents of this address &FCC1 are to be output (rather than input) and this is achieved by the instruction:

$?\&FCC3 = 255$ in 'Basic'

or $LDA \# 255 : STA \&FCC3$ in 'Assembler'

This instruction is always needed before the Interface is used so it is a good idea to make it the first operation in any Interface program. Set up need only be done once so there is no need to repeat it if the Control Code is changed.

Control code

Once the Interface is ready to receive the Control Code, it can be given the code itself. Pages 16 & 17 show the possible Control Codes. Setting of the Control Code is a two stage operation. First of all, the Control Code proper is sent to the Interface:

e.g. $?\&FCC1 = 32$ in 'Basic'

or $LDA \# 32 : STA \&FCC1$ in 'Assembler'

Direction code

The Interface must now be told whether its data lines are going to be inputs or outputs, i.e., whether it is going to take *in* information for feeding to the computer or take information from the computer to feed *out*. Usually this Direction Code either sets all the data lines to inputs or all the data lines to outputs but there are occasions when a combination of inputs and outputs are used. The Direction Code is put into location &FCC2.

If all the data lines are to be inputs (such as when taking in analogue information or when using all the digital lines as inputs), the instruction will be

```
?&FCC2 = 0 in 'Basic'  
  
or LDA # 0: STA &FCC2 in 'Assembler'
```

If the D to A output is to be used or the digital input/output lines are all to be outputs, the instruction will be

```
?&FCC2 = 255 in 'Basic'  
  
or LDA # 255: STA &FCC2 in 'Assembler'
```

For a combination of inputs and outputs, a Direction Code is selected such that a '1' is placed in each binary position required as an output and a '0' for each binary position required as an input. To save you time and trouble, the Direction Codes for all the possible combinations are given alongside the Control Codes on Pages 16 and 17.

Experienced users will recognise that the SET UP and DIRECTION CODE instructions are involved with the setting up of Data Direction Registers in the input/output device inside the Interface (see Page 34).

NOTE: Perhaps the best way of understanding the function of the control and direction codes is to inspect the listing of a simple program to identify their occurrence and use. For example 'RLYOPT' demonstrates the use of a relay followed by an input of data through an analogue input. "RLYOPT" is described on Page 38.

5.2 CONTROL AND DIRECTION CODES

Once the user has decided how the Interface is to operate and he has entered the SET UP command:

```
?&FCC3 = 255
```

the appropriate Control and Direction Codes must be selected and entered. These Codes can be entered either 'direct' via the keyboard or as part of a program in the computer. In many instances, the Interface may be required to operate in different ways at different stages and in these cases the Control and Direction Codes may be changed at various points in the program.

Control and direction codes for analogue input

There are 32 different ways in which the analogue input system can work and the mode is selected by means of a Control Code between 0 and 31 inclusive. In all instances, the Direction Code for analogue input is 0, so one would enter the following 'direct' or as part of the program:

```
?&FCC2 = 0 in 'Basic'  
  
or LDA # 0: STA &FCC2 in 'Assembler'
```

The Control Code is then selected from the following table:

analogue input codes				
input				
A	B	C	D	range
0	1	2	3	0 to + 10v
4	5	6	7	0 to + 1v
8	9	10	11	0 to + 100mv
12	13	14	15	variable + only
16	17	18	19	-5 to +5v
20	21	22	23	-0.5 to +0.5v
24	25	26	27	-50 to +50mv
28	29	30	31	variable - & +

Fig. 11

(See Page 20 for further information on the analogue input system). Thus, for example, if you wish to read in analogue information to input 'A', with a sensitivity of 0 to +1 volt, the Control Code would be entered as follows:

```
?&FCC1 = 4 in 'Basic'

or LDA # 4: STA &FCC1 in 'Assembler'
```

Control and direction codes for digital inputs and outputs

State of Each Digital Line						Direction Code		Control Code
7	6	5	4	3	2	1	0	
In	In	In	In	In	In	In	In	32
In	In	In	In	In	In	Out	Out	64
In	In	In	In	Out	Out	Out	Out	96
In	In	Out	Out	Out	Out	Out	Out	128
Out	Out	Out	Out	Out	Out	Out	Out	160

Fig. 12

For example, if lines 1, 2, 3 and 4 were needed as outputs and lines 5, 6, 7 and 8 as inputs, the following would be entered:

```
?&FCC2 = 15 : ?&FCC1 = 96 in 'Basic'

or LDA # 15 : STA &FCC2 : LDA # 96 : STA
&FCC1 in 'Assembler'
```

Control and direction codes for relay operation

The relays reflect the state of digital lines 0, 1, 2 and 3 when a Control Code of 192 is brought into operation. This Control Code establishes lines 0, 1, 2 and 3 as outputs as might be expected but it also sets lines 4, 5, 6 and 7 as inputs. The Control Code 192 lifts the latch on the relays so that they can be changed if required. To drop the latch and therefore 'lock' the relays in the position selected, the Control Code is changed to any number other than 192. The Interface can then go to read off

analogue or digital inputs without the relays being affected by data received on these inputs. The Direction Code to use when in relay movement operations is 15.

```
?&FCC2 = 15 : ?&FCC1 = 192 in 'Basic'

or LDA # 15 : STA &FCC2 : LDA # 192 : STA
&FCC1 in 'Assembler'
```

Control and direction codes for analogue output operation

As with the relays, the analogue output is latched. This latch is lifted by entering a Control Code of 224 after which the analogue output will depend on the data on the eight digital data lines. As soon as the Control Code is changed to some other value, the latch will drop and the analogue output will remain at the value last reached before the

latch was dropped. The corresponding Direction Code to use is 255.

```
?&FCC2 = 255 : ?&FCC1 = 224 in 'Basic'

or LDA # 255 : STA &FCC2 : LDA # 244 : STA
&FCC1 in 'Assembler'
```

5.3 DELAYS

In almost every data collection situation, it is necessary to time the data collection process in some way. If the data collecting process is relatively slow, it is likely that the program will operate in 'Basic' but if rapid collection is required, the program must be written in 'Assembler' for machine code operation at the required speed. Whichever is used, the program will have to be able to operate at a speed higher than that required by the data collection process as otherwise it is likely some data will be lost.

The outcome of this is that nearly all programs, whether written in 'Basic' or 'Assembler', will have to include some delay or timing routines which can be used between each data collection cycle.

Delays in 'Basic'

Computer users will be familiar with the oft met routine in a program:

```
FOR N = 0 TO 2000 : NEXT N
```

This is a delay routine which is simple to vary in length but has the disadvantage that the length of the delay incurred can not be simply evaluated. As an alternative, the computer's own 'Basic' clock can be used. This increments 100 times per

second and can be set to zero at the start of a program or after a sample has been taken. It is best not to set the clock to zero after each sample since zeroing takes a finite time and this time will not be accounted for in the data sampling process. The following routine can be included in any 'Basic' program to provide the delay you require. Your program, sampling the data, will fit between the clock setting line 10 and the subroutine at line 10000 and onwards:

```
10 VDU12:PRINT "What time delay do you
require (seconds) ":INPUT DELAY:CENTI=DELAY*
100:TIME = 0 : A=0
```

```
550 REM CALLS DELAY
560 GOSUB 10,000
```

```
10000 A=A+CENTI
```

```
10010 REPEAT
```

```
10020 UNTIL TIME >= A
```

```
10030 RETURN
```

To use the delay within the program, line 10000 onwards is called as a subroutine. Your program will call the delay when required, e.g.,

```
50 GOSUB 10000
```

Delays in 'Assembler'

Machine code delays can be called as a subroutine but the process is rather more complicated especially if lengthy delays are required. The technique is based on the timers of the in/out chip contained in the Interface together with one zero page memory location. Since low rate sampling can be conducted using a program in 'Basic', there is no need to provide the facility for low rate sampling in a machine code routine

and the maximum delay of approximately 16 seconds in this routine should prove more than adequate. At some point in your program, before sampling takes place, the computer has to be told what the delay is to be and from this information it will compute the values of the delay variables A, B and M. Thus the program will contain the following:

```
5 DIM HOLD 100:REM space for code
10 VDU12:INPUT "" "What delay is required? (microseconds) "T
15 IF T<25 OR INT(T/5)<>T/5 PRINT"" "Delay must be at least 25 microseconds
and a multiple of 5 microseconds." "" "Press 'space-bar' to continue.":A$=GET$:GO
TO10
20 IF T>16777215 PRINT"" "Delay cannot be larger than 16777215 microsecond
s." "" "Press 'space-bar' to continue.":A$=GET$:GOTO10
50 T=T-25
60 M=T DIV 65536
80 R=T-(M*65536)
90 B=R DIV 256
100 A=R MOD 256
```

The machine code delay routine itself has to be assembled within your BASIC program. This would be written as follows:

```
500 REM machine code is now assembled
510 FOR X = 0 TO 2 STEP 2
520   P%=HOLD
530   [OPT X
550   .DELAY SEI:NOP:NOP:LDA #M:STA&70
560   LDA &70
570   BEQ LABEL1
580   .LABEL2 LDA #255
590   STA &FCC4
600   STA &FCC5
610   .LABEL3 LDA &FCCD
620   AND#64
630   BEQ LABEL3
640   DEC&70
650   BNE LABEL2
660   .LABEL1 LDA#A
670   STA &FCC4
680   LDA#B
690   STA &FCC5
700   .LABEL4 LDA &FCCD
710   AND #64
720   BEQ LABEL4
730   CLI:RTS
740   ]
750 NEXT X
```

When the time delay is required in your program, you insert the instruction:

JSR DELAY

The delay subroutine can be called whenever you wish during your program. You can even call the machine code delay in 'Basic' by using:

CALL DELAY

Notice that the delay routine has a *minimum* time of 25 microseconds. It is unable to offer delays of less than this. Again, because of the structure of the routine, it can only provide delays which are multiples of 5 microseconds. The maximum

number of microseconds which can be handled is 16,777,215 (i.e. a delay of 16. 777215 seconds). This is produced when the variables M, B and A all contain the number 255 as a result of the calculations performed on the input delay time, as in lines 60 to 100.

The accuracy of the delay produced is better than 1 μ s for delays of 65535 μ s and less. For delays larger than this, the accuracy is of the order of $\pm 7 \mu$ s.

Delays of Different Periods

In some applications, you may need delays of different duration during one program. The simplest way to do this is to call the DELAY subroutine again, having inserted different values into M, B and A.

5.4 THE ANALOGUE INPUT SYSTEM

The analogue input system provides the means to connect up to four transducers to the Interface. These four input signals can be read in any order and although only one input can be read at a time, the time needed to read an input, switch to and read a second input is extremely short, being of the order of a few microseconds.

Besides being able to select information from any one of four input channels called A, B, C and D, the Interface can assign to any channel any one of eight different voltage multipliers.

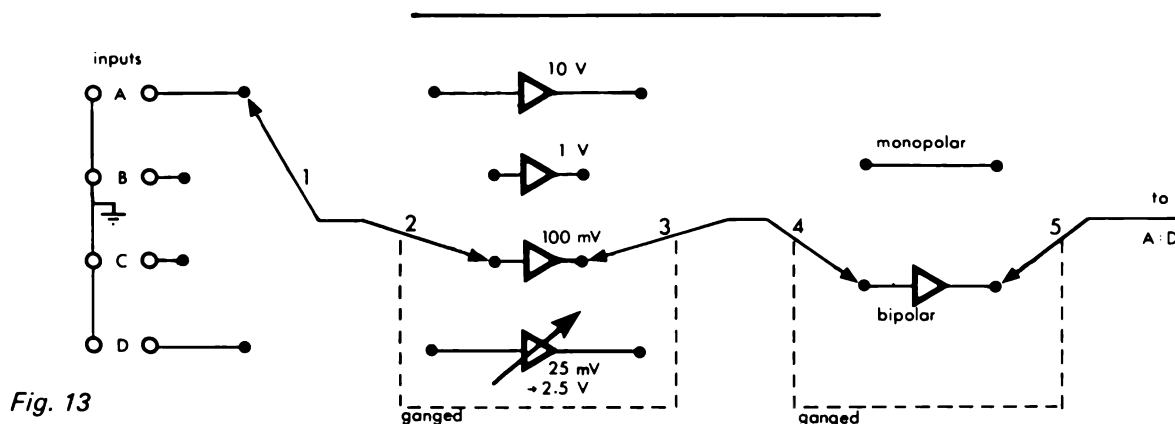


Fig. 13

The switches 1, 2, 3, 4 and 5 are electronic switches which are controlled by the computer via the control codes. Rather than "click the switches round" as would happen if they were operated manually, the computer can operate them all simultaneously in one step.

SWITCH 1 – selects which input is being read

SWITCH 2 AND 3 – are ganged. They select the required multiplier to give full scale sensitivities of:

- 10 volts
- 1 volt
- 100 millivolts
- variable 25 mV to 2.5V

SWITCH 4 AND 5 – are ganged. In the first position, the signal is fed through to the A to D converter unchanged. In the second position the signal is fed to the bipolar amplifier which converts an incoming positive and negative signal into a positive only signal. This is achieved by adding to the incoming signal, whatever its value, an amount equal to half full scale sensitivity.

This is equivalent to shifting the zero line down:

e.g., system is on 10 volt range. Switch in the bipolar amplifier which effectively adds 5 volts ($\frac{1}{2}$ full scale sensitivity) to the incoming reading. The unit is now ready to accept values between -5 and $+5$ volts. On receiving $+1.3$ volts, it converts it to $(1.3+5)$ 6.3 volts. If the incoming value is -3.4 volts it will be converted to $(-3.4+5)$ 1.6 volts. If necessary the resulting positive bias can be corrected in the computer program so that true voltage levels can be read into the computer memory.

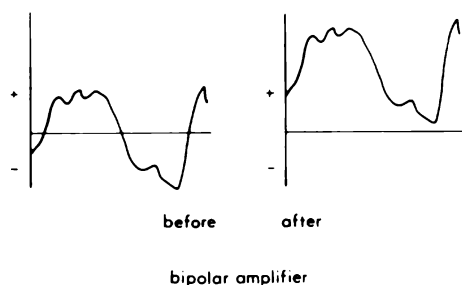


Fig. 14

Analogue input - an example

Possibly the foregoing may seem somewhat puzzling, in which case it will be a help if we try a simple example.

This first program will make the Interface take readings on the analogue input A and display the

results on the screen. You need to connect some variable voltage supply, maximum 9 volts, to input A. A 9 volt battery to a high resistance (1K) potentiometer is ideal.

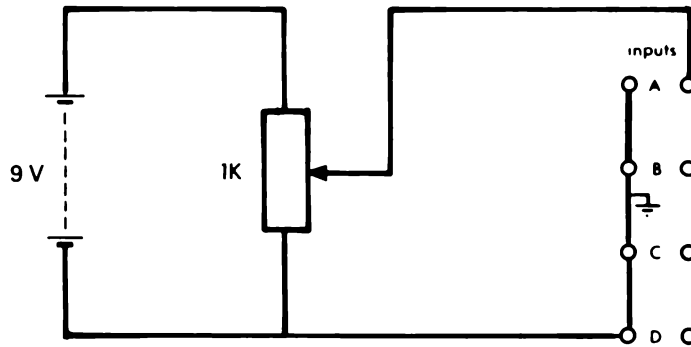


Fig. 15

```

10 REM First of all, SET UP
20 ?%FCC3=255
30 REM Now enter the DIRECTION CODE for analogue input
40 ?%FCC2=0
50 REM Clear the screen
60 VDU12
70 REM Find out the CONTROL CODE to be used
80 INPUT""What is the CONTROL CODE you wish to use"CONTROLCODE
90 REM Apply the control code
100 ?%FCC1=CONTROLCODE
110 REM Take a reading from the data address
120 AREADING=?%FCC0:PRINTAREADING
130 GOTO80

```

This program is included in the text so that it can be studied as we work through the development of an analogue input program. Now follow the instructions step by step:

1. RUN the program and after setting up the Interface and applying the DIRECTION CODE, you are asked which CONTROL CODE you wish to use. Initially we are using only analogue input A and, if you turn to Page 17 you will see that:

For A to read	The Control Code is
0 to 10 volts	0
0 to 1 volt	4
0 to 0.1 volts	8
0 to 25 mV - 0 to 2.55 volts	12
-5 to +5 volts	16
-0.5 to +0.5 volts	20
-50 mV to + 50 mV	24
-12.5 to +12.5mV - -1.25 to +1.25 volts	28

To start with, enter a CONTROL CODE of 0, giving an input sensitivity of 0 to 10 volts.

2. On pressing return, the reading from the analogue input is presented on the screen.
3. Repeat the operation with the same CONTROL CODE but note the effect of changing the input voltage.
4. Turn the input voltage down to zero and run the program again with a CONTROL CODE of 4, thus increasing the sensitivity of the A input to 0 to 1 volt.
5. Repeat step (4) but with increasing values of input voltage. Note that the reading as displayed on the screen will increase to a maximum value of 255.
6. Notice that at this point the level indicator on the interface illuminates. This indicator will light whenever any analogue input maximum is reached. At that point, the A to D converter is producing its maximum output of 255 and it cannot go any higher.
7. You can try running the Interface with a CONTROL CODE of 8 but it will be difficult with the battery/potentiometer suggested to avoid overloading the input. Such an overload will not damage the unit. (Turn to Page 31 for further information about the internal circuitry protection fitted to the Interface).
8. The next CONTROL CODE, 12, brings in the variable gain amplifier which can, by means of the Amplifier gain control on the Interface, be set to any sensitivity between 25 mV and 2.5 volts full scale. You will now find that the reading displayed on the screen is a function of both the input P.D. and the setting of the

Amplifier gain control. When the variable gain amplifier is selected, the gain potentiometer can be adjusted so that the LED (light emitting diode) is just off when the maximum voltage is input.

9. Trying next a CONTROL CODE of 16, you will find that a zero input results in a displayed reading of 128 which is half the maximum value. Positive inputs will result in this reading increasing but if the input P.D. is reversed the reading indicated will fall below 128, reaching 0 at an input P.D. of -5 volts. Thus the 0 to 10 volt range when the CONTROL CODE was 0 has been changed to a -5 to +5 volt range when the CONTROL CODE is 16.
10. Now that the means for changing input sensitivity has been tried and tested, we can apply a little polish to the program. At present, the unit always reads on a scale of 0 to 255. In many instances we need to obtain actual values of inputs and we can use the computer to do this for us. In the following example, the input sensitivity is set to 0 to 10 volts and the input reading is converted to give actual input voltage.

Change line 120 to read:

```
120 AREADING=?&FCC0:PRINT
(AREADING* 10/255)
```

11. RUN the program with a CONTROL CODE of 0 and you will find that the system now operates as a 0 to 10 volt voltmeter. Before you collapse in amazement at the magnificent achievements of this modern technology, read on!

5.5 AUTORANGING

The ability of the computer to change the CONTROL CODE on the basis of the data it receives, enables the voltmeter to be autoranging. If a reading is too large, the computer can switch to a less sensitive input amplifier and scale the readings accordingly. Alternatively when the reading is very small and the resolution is therefore rather poor, the computer can instruct the Interface to switch to a more sensitive range with appropriate rescaling of the reading.

Enter the program into the computer - Page 37 "ANALOG2"

When RUN, this program will operate the system as a continuously reading voltmeter which is autoranging and which also gives a display of its resolution. Remember that an open circuit input will introduce mains hum which will produce a non-zero indication.

As it stands, this program is little more than an illustration of potential but the ability to autorange when data recording is important.

5.6 THE INTERFACE AS DATA RECORDER

One very important use for the Interface is in data recording. The computer provides a very large quantity of memory, much of which can be made available for storage of data. The high speed of the Interface enables the user to record rapidly changing data then to process it, plot it on the screen or replay it slowly to a printer or chart recorder. Alternatively, the Interface and computer can be used to record data relatively slowly over very long periods of time.

Once the data has been collected, there are several things that can be done with it:

- (a) It can be plotted as a graph using high resolution graphics. Values can, if required, be converted into some other form before plotting. For example, the 'y' variable could be plotted against the logarithm of the 'x' variable.
- (b) Values collected may be totalled thus effectively measuring the area under a graph.
- (c) Values can be group averaged before plotting to minimise the effects of unwanted noise.
- (d) The data can be replayed slowly to a chart recorder or a printer.
- (e) The data can be 'dumped' in large blocks on to a cassette tape or a disc ready for later examination.
- (f) Data can be allocated to 'bins' according to magnitude and subsequently plotted in histogram form showing the frequency with which certain ranges of values were encountered. Other statistical procedures can be applied to the raw data.

"GENALOG" is a multipurpose program providing many of the above facilities. The program can be used as it stands as a general purpose data collection, handling and plotting program. Its main purpose, however, is as an illustration of techniques which you can tailor to your own purposes in your own programs.

The most complex operation in "GENALOG" is that involved in the production of accurate time delays between the taking of each sample. If the Interface is to make best use of its ability to take readings very rapidly, the data collection must be achieved in a machine code routine written via the computer's Assembler. The timing routine uses the timer in the Interface which is limited in this instance to a maximum delay of about 16 seconds, and a minimum delay of 40 microseconds.

Enter the program into the computer - page 37 "GENALOG" or "GNLGCSS".

Using the program

When the program is RUN, you will be asked a series of questions relating to the operating conditions you require:

1. Which Input do you wish to use? A, B, C or D.
2. Which input amplifier do you wish to use?
0 to 10 volts
0 to 1 volt
0 to 100 millivolts
Variable 0 - 25 mV to 0 - 2.5 volts.
3. Will the signal ever go negative?
4. Do you want sampling to start on the trigger?
If so, is the trigger to operate on a positive going or negative going signal?
5. What delay do you require between each sample?
6. How many samples do you wish to take?
(option not given if high res. graphics chosen; 250 samples taken).

In the above sequence, questions (1), (2) and (3) are used to find out the CONTROL CODE to be used. If the answer to (3) is yes, the bipolar amplifier will be brought into operation and as a result, the amplifier range specified in (2) will be centred about 0 volts. For example, if the 0 to 10 volt amplifier was selected in (2), a yes answer to (3) will make this into a -5 to +5 volt range. Question (4) determines the mode of operation of the trigger circuit (see page 24). After Question (6) has been answered, the trigger level should be adjusted if necessary and the system will be ready to start its task.

Sampling Data from more than one Analogue Input

The program needed to perform this operation is very straightforward. The sequence of operations is:

- (a) SET UP
- (b) Enter the DIRECTION CODE
- (c) Enter the first CONTROL CODE
- (d) Read the DATA from the first input
- (e) Change the CONTROL CODE
- (f) Delay if necessary

- (g) Read the DATA from the second input
- (h) Change the CONTROL CODE
- (i) Delay if necessary

... and so on.

An example of such a program, "4INPUT", is given on Page 37.

This program takes readings from four 0-100 mV input sources in sequence, giving information on light level, pH, temperature and oxygen level.

After samples have been taken from each input, the results are plotted as four separate graphs. Such a program has applications for the environmental scientist who wishes to monitor certain aspects of a locality such as a pond or stream. In such a case the speed of this demonstration program would need to be reduced considerably by including a delay which can be tied to the computer clock as on Page 18.

5.7 THE TRIGGER

The trigger system on the Interface is used to initiate the input or output of a piece of data. In practice, the trigger is usually used to tell the Interface and Computer when to take in an analogue reading. Trigger operation is controlled in software so there is considerable flexibility in its use.

The trigger can be operated by either a positive going or negative going signal. It can be used to initiate every reading the Interface passes to the computer or it can be used to start off a block of readings.

The trigger is controlled by means of the number stored at location FCCC (Hex). A value of 0 gives falling edge triggering whilst 16 gives rising edge triggering. To make the trigger operate on a rising input, enter the following routine in your program immediately before the instruction to read the data:

```
In 'Basic'    ?&FCCC = 16
              REPEAT
              TRIG = ?&FCCD AND 16
              UNTIL TRIG = 16
```

```
In 'Assembler' LDA # 16: STA FCCC
               .LABEL LDA & FCCD
               AND # 16
               BEQ LABEL
```

For trigger operation on a falling input, the following routine should be placed immediately before the instruction to read the data:

```
In 'Basic'    ?&FCCC = 0
              REPEAT
              TRIG = ?&FCCD AND 16
              UNTIL TRIG = 16
```

```
In 'Assembler' LDA # 0: STA &FCCC
               .LABEL LDA & FCC0
               AND # 16
               BEQ LABEL
```

If you wish to initialise every reading through trigger operation, the routine above should be placed in the program as a subroutine which is called immediately before each data sample is read.

In all cases, the trigger is cleared by reading from or writing to location FCC0, the data address. It is a wise precaution to clear the trigger as a matter of course in the program before the data reading section otherwise the first sample may be taken immediately without waiting for the next trigger pulse. To clear the trigger, simply read the data address. There is no need to use the information so obtained.

TRIGGER SENSITIVITY

The level at which the trigger operates is set by means of the trigger sensitivity control on the Interface control panel. To assist in the setting of this control, an indicator is provided beside the trigger sensitivity control. This indicates the level on the trigger line. To set up the trigger, it is usual to try a dummy run with the program and feed in the signal level at which the trigger is to operate. The control is then adjusted so that at this level the indicator just lights if the trigger is to operate on a rising edge. If triggering is required on a falling edge, the trigger sensitivity is adjusted so that the indicator just goes out at the required signal level.

As an example of the control of sampling using the trigger, a program is included which takes 256 samples at the maximum possible rate of 125,000 samples per second. The sampling starts on the arrival of the transient to be sampled.

This program is "FLATOUT", on page 37.

5.8 THE ANALOGUE OUTPUT

The analogue output can provide a P.D. output in the range of 0 to 2.55 volts or a current output of 0 to 1 mA, corresponding to data values of between 0 and 255. The analogue output is latched and can only be changed if the latch is lifted by applying a CONTROL CODE of 224 together with a DIRECTION CODE of 255.

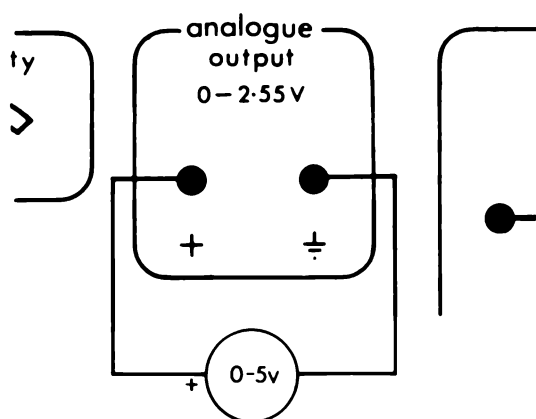


Fig. 16

Connect a 0 to 5 V voltmeter or a 0 to 1 mA meter to the analogue output as shown in Fig. 16. Enter the following into the computer:

```
10 ?&FCC3 = 255 : REM Set up
20 ?&FCC2 = 255 : REM The Direction Code
30 ?&FCC0 = 0 : REM Data set to zero
40 ?&FCC1 = 224 : REM The Control Code for D
    to A operation.
```

When RUN, the analogue output voltage should be 0 volts (0 mA).

Try altering the data in FCC0 to different values:

```
e.g. ?&FCC0 = 25
      ?&FCC0 = 120
      ?&FCC0 = 255
```

and you will see that the analogue output moves to corresponding values (if your voltmeter readings appear low, you may be using a voltmeter with an input impedance which is too low).

Now try altering the CONTROL CODE before changing the data again:

```
?&FCC1 = 150
?&FCC0 = 0
```

The analogue output P.D. will not change because changing the CONTROL CODE 'drops' the analogue output latch. As with the latch on the relay operations, this is a very important facility which enables the interface to switch to measure an input while maintaining an output P.D.

Please note that it is important that the CONTROL CODE is changed from 224 before the DIRECTION CODE is changed as otherwise the analogue output may change value before the latch is dropped.

Chart Recorder Output

Very often it is desirable to be able to plot data as a permanent record on a chart recorder. The Interface is particularly useful in this application because it can be used to temporarily store rapidly changing data then play it out at a rate suitable for the relatively slow response rate of the chart recorder. Between record and replay of data, the information can be manipulated in a wide variety of ways, including averaging to reduce noise.

The following routine will output data from the computer memory to a chart recorder connected to the analogue output. Ideally, the chart recorder should have a full scale sensitivity of 2.55 volts and it may need to be adjusted to this sensitivity with an attenuator. The data that you wish to pass to the chart recorder will be stored somewhere in the computer memory. In this listing the lowest and highest memory location are referred to as (lowest) and (highest). You will need to substitute the actual locations in place of these names before you use the program.

```
1000 ?&FCC3 = 255 : REM Set Up
1010 ?&FCC2 = 255 : REM Direction Code
1020 ?&FCC1 = 224 : REM Control Code
1030 FOR N = (lowest) TO (highest) : REM
    Insert appropriate locations.
1040 FOR X = 0 TO 200 : NEXT X : REM Delay
    to slow down the data output rate.
    Adjust this to suit your chart recorder.
1050 A = ?N
1100 ?&FCC0 = A : REM Output the data
1200 NEXT N : END
```

If you wish to adjust the data in some way, perform the operations on the variable A and use the line numbers between 1050 and 1100. You could for example:

- (a) Subtract a constant from A so moving the zero level upwards.
- (b) Multiply each value by a constant to exaggerate small differences.
- (c) Output the logarithm of the value stored.

Programs which include good examples of the use of the analogue output are "ECG" and "BALANCE" on Pages 38 and 39 respectively.

Waveform Generation

The analogue output can be used to provide very complex waveforms having up to 256 different amplitude levels.

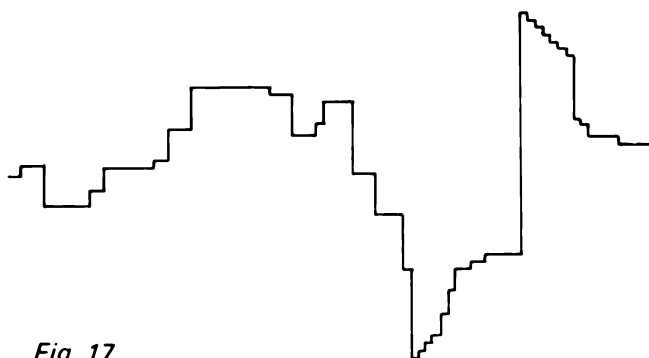


Fig. 17

Each successive amplitude level is stored as data in successive memory locations in the computer. These levels are read out to the analogue output. When the last memory location has been read, the program jumps back to the first location again to repeat the cycle. This operation needs to be performed rapidly so the program must be written in machine code via the Assembler

SECTION 6

CONTROL

There are three ways in which the Interface can control an external load:

- (a) By means of the 4 relays.
- (b) By the use of digital outputs operating external drivers.
- (c) Through the D to A output.

6.1 THE RELAYS

Each of the four relays contains a single pole changeover contact capable of switching a load of 1 amp at up to 24 volts. If the load exceeds either of these ratings, it is recommended that the Interface relays be used to operate external heavy duty relays or an appropriate power transistor circuit.

When brought into operation, the relays marked 0, 1, 2 and 3 reflect the logical state of the digital lines 0, 1, 2 and 3 respectively.

The relays are under latch control, operated by use of the CONTROL CODE 192. When a CONTROL CODE of 192 is entered, the state of the relays can be changed at will according to the binary value put out to the data address FCC0.

When the CONTROL CODE is changed to some other value, the relays will stay in their last set position.

Whilst CONTROL CODE 192 is in operation, digital lines 0, 1, 2 and 3 will operate as outputs and digital lines 4, 5, 6 and 7 will operate as inputs. Before the CONTROL CODE for relay operation is applied, the DIRECTION CODE must be set to 15.

Relay control can best be understood by trying some examples. To start with, try the following program:

```
10 ?&FCC3 = 255 : REM Set Up
20 ?&FCC2 = 15 : REM The Direction Code
30 ?&FCC0 = 0 : REM Data set to zero
40 ?&FCC1 = 192 : REM Control Code for relay operation
```

When you RUN this program you may hear a click as relays switch off. This short program sets up for relay operation but also puts a data value of 0 out to the relays so switching them all off. The remaining operations will be more easily observed if the relays are used to operate lamps. Connect a battery and bulb in series with each of the relay contacts as shown in Fig. 18.

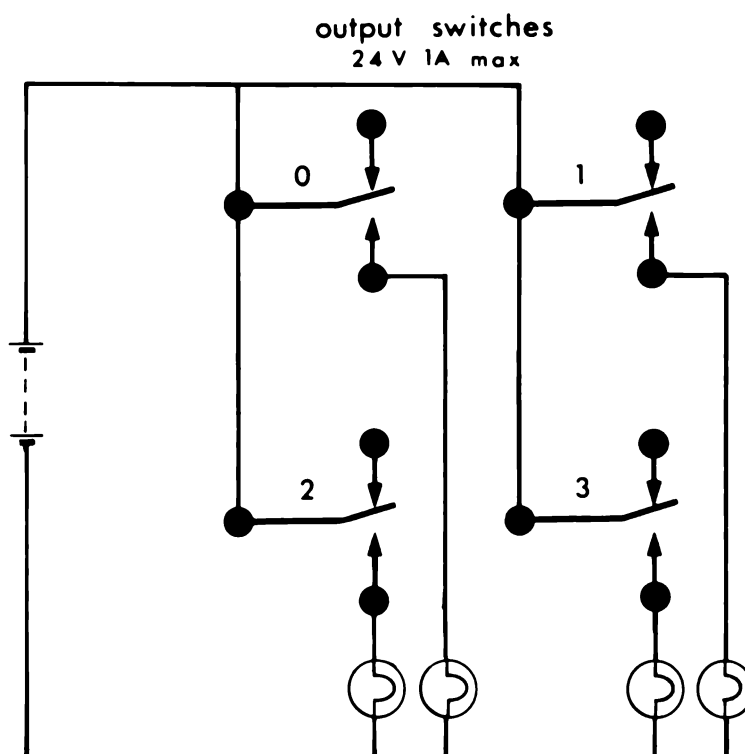


Fig. 18

We have left the CONTROL CODE 192 in operation so we can operate the relays from the keyboard. Try the following:

- ?&FCC0 = 1 Relay 0 should operate because
1 is binary 00000001
- ?&FCC0 = 0 Relay 0 switches off because
0 is binary 00000000
- ?&FCC0 = 8 Relay 3 should operate because
8 is binary 00001000
- ?&FCC0 = 9 Relays 0 and 3 should operate
because
9 is binary 00001001

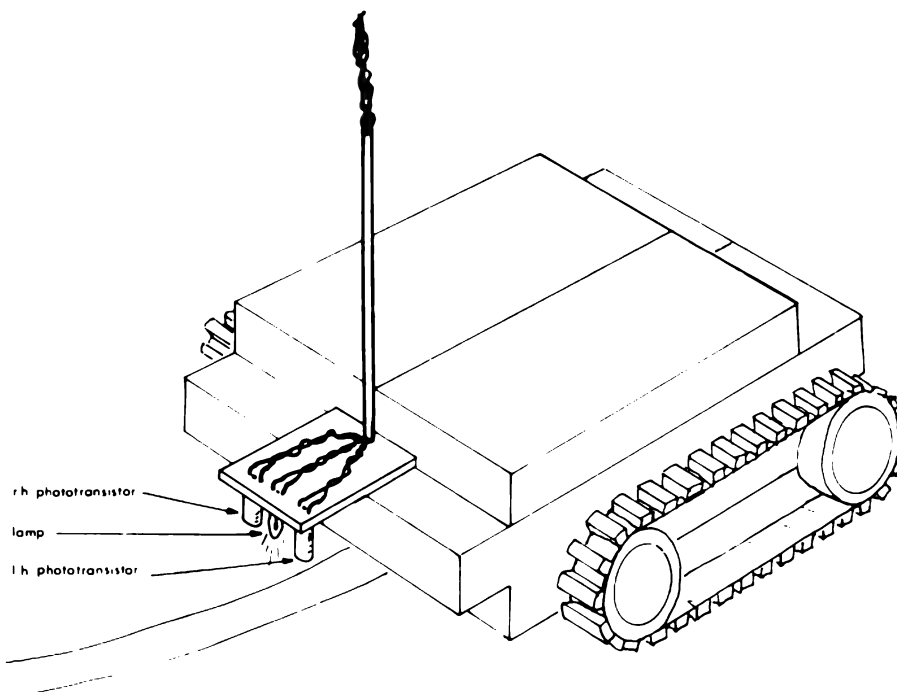
Change the CONTROL CODE to, for example, 0:
?&FCC1 = 0

Try changing the data now and you will find you can not change the relays. They have been latched on to their last state with relays 3 and 0 switched on.

To make the Interface do some useful control, we need to make the relays operate on the basis of information received through an input.

6.2 WHITE LINE FOLLOWER

Fig. 19



This unit has been chosen as an early example in computer control because it is spectacular to watch but relatively simple to implement. Each side's wheels can be driven independently of those on the other side. A very simple way to do this is to use two Lego motors fixed together. The wheels are part of the Lego Motor Kit but each motor is fitted with wheels on one side only.

At the front of the car, a small lamp is fitted, with a phototransistor at each side facing downwards. The phototransistors must be fitted inside small

black tubes so that the field of view of each one is restricted to a small part of the road underneath. Some experimentation is needed with the brightness of the light to obtain optimum results. The phototransistors are connected to digital inputs 5 and 6 as shown in Fig. 20, the left hand phototransistor being connected to input 6.

The leads which provide power to the motors and lamp and which bring the feedback signal from the phototransistors to the Interface should be very flexible and supported from above the car.

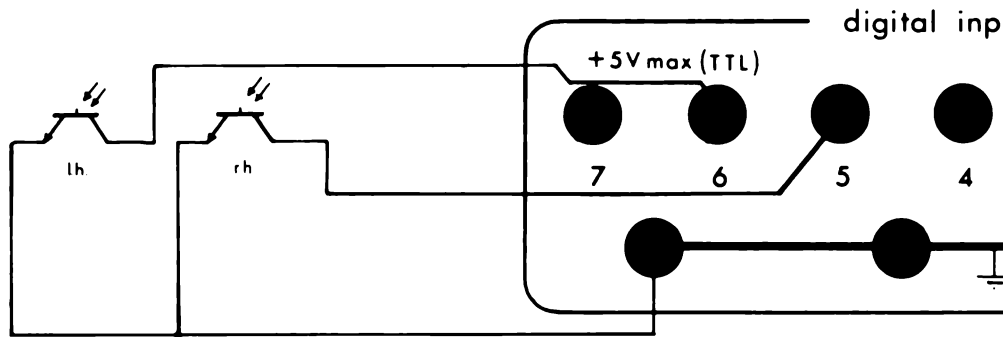


Fig. 20

Before the unit will operate, seven lines must be added to the program on Page 27:

```

50 L = ?&FCC0 AND 64 : REM L becomes 64 if
   photodetector is in darkness, otherwise L =
   0
60 R = ?&FCC0 AND 32 : REM R becomes 32 if
   photodetector is in darkness, otherwise R =
   0
70 IF L = 0 AND R = 0 THEN ?&FCC0 = 3 : REM
   Car is on the white line
80 IF L = 64 AND R = 0 THEN ?&FCC0 = 2 : REM
   Car is too far left of the white line
90 IF L = 0 AND R = 32 THEN ?&FCC0 = 1 : REM
   Car is too far right of the white line
100 IF L = 64 AND R = 32 THEN
   VDU12:PRINT""""I AM LOST"" :?&FCC0=0:
   END
110 GOTO 50 : REM Go back and check position
   again.
```

Line 50 checks to see if the left hand detector can see the white line and line 60 checks the same for the right hand detector. Line 70 puts on both the left and right hand motors, via relays 1 and 0 respectively, if both detectors can see the white line. If only the right hand detector can see the line, line 80 switches the left hand motor on and the right hand motor off, so swinging the car to the right and, hopefully, back on to the white line. Line 90 corrects similarly for a position too far to the right of the line. Line 100 is used when the car has lost the line completely.

A point to remember here, is that when using the relays, a Direction Code of 15 ensures that the top 4 digital lines act as *inputs*. So the data port

&FCC0 can be *read* (as in lines 50 and 60) for information about the phototransistors, and then *written to* (as in lines 70 to 100) to alter the relays, without changing the Direction and Control Codes.

The white line should be laid in a loop on a flat, matt black surface. White PVC tape works well. The line must not be too thin and ideally needs to be rather broader than the distance between the two phototransistors. It is advisable to include some type of fine control on the motor voltage as the speed of the motors can be quite critical if the tendency of the car to over-correct its direction of movement is to be avoided.

The power and versatility of software control can be very ably demonstrated by making a few minor changes to the program used so far. Change lines 70 to 100 as follows:

```

70 IF L = 64 AND R = 32 THEN ?&FCC0 = 3
80 IF L = 64 AND R = 0 THEN ?&FCC0 = 1
90 IF L = 0 AND R = 32 THEN ?&FCC0 = 2
100 IF L = 0 AND R = 0 THEN VDU 12 : PRINT """"I
   HAVE ERRED AND STRAYED...
   HELP !"" :?&FCC0 = 0:END
```

The car will now operate as a white line avoider! Placed inside the loop of white tape, the car will move around, changing direction whenever it meets the tape.

NOTE Refinements to the program may be necessary, depending on the characteristics of the particular model, e.g., the level of illumination from the reflected light.

SECTION 4

ADVANCED TIMING APPLICATIONS

4.1 MACHINE CODE TIMING

Operations in 'Basic' are executed relatively slowly by a microcomputer. Whilst in the previous programs, the degree of precision may be sufficient for most purposes, timing exercises in 'Basic' do not make fullest use of a computer's ability to time events to great precision.

The Unilab Interface contains an Input/Output (I/O) chip with timers. These timers can time in intervals of 1 microsecond with the precision of the quartz clock crystal inside the microcomputer.

To take advantage of the high counting rate of these clocks, the timing part of the program needs to be executed in machine code. The simplest way to enter this code is to use the Mnemonic Assembler in the microcomputer. If you have not used an Assembler before, the thought may seem rather daunting. For this reason, we have provided a number of routines which the user can copy and use as needed without necessarily understanding how they work. The more advanced user may wish to turn to Page 34 which provides factual information about the input/output circuitry in the Interface.

There is one important point that has to be remembered when using software timing. To detect a change of an input, the software runs the microprocessor round a loop of instructions. This loop takes a finite amount of time to execute, typically between 7 and 50 microseconds depending on its length. Within this loop there is only one point when the microprocessor 'reads'

the input to see if it has changed. Thus there may be some delay in the timer starting. The size of this delay will vary according to the point the microprocessor has reached in the loop when the input changes. Similarly there will be a range of possible delays which may be incurred through the loop at the end of the timing period. Whilst for most of the timing period, the time is measured at high resolution and accuracy, the loops at each end of the timing routine produce an error which can be significant if the timing period is extremely short. Figure 8 shows this diagrammatically.

These errors are minimised if the time taken to circulate each program loop is kept as short as possible. These loops can be very short provided the microprocessor does not need to look for the 'timing out' of a timer. The timer inside the Interface will count to a maximum of 65536 microseconds, i.e., approximately 65 milliseconds. By using a 'time out flag', this maximum measurable period can be doubled. If periods greater than about 130 milliseconds are to be timed, one needs a more complex program with longer loops. Such a program will note how many times the timer timed-out during the 'on' period and will take this into account when calculating the total elapsed time.

Two forms of machine code timing program are given. The first will time periods of up to 131072 microseconds with an error of ± 15 microseconds. The second program will time periods of up to rather more than 4000 seconds but the error is ± 60 microseconds.

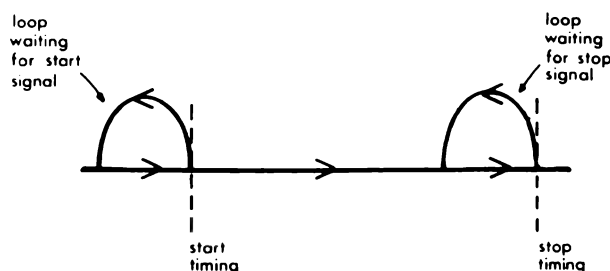


Fig. 8

SECTION 7 INFORMATION

7.1 SPECIFICATION

Analogue Inputs

Input resistance 1 Megohm

Conversion rate of 125,000 per second.

Sensitivity 0 to +10 volts

0 to +1 volt

0 to 100 millivolts

variable 25 mV to 2.5 volts f.s.d.

–5 to +5 volts

–0.5 to +0.5 volts

–50 to +50 millivolts

variable –12.5 to +12.5 mV up to –1.25 to +1.25 volts

Analogue Output

0 to +2.55 volts through 2.45 Kohms to give

0 - 1 mA into a 100 Ω load.

Setting time of 1 μ s

Digital Inputs

Up to 8 digital inputs of TTL type.

Digital Outputs

Up to 8 digital outputs of TTL type.

Relay Outputs

Four single pole changeover relays with contacts rated at 1 amp, 24 volts.

Trigger

Linked to the analogue input system can be used to initiate action on a rising or a falling analogue input.

7.2 ELECTRICAL PROTECTION

It is very important that a computer interface provides protection for itself and for the computer to which it is connected. There is always a danger that a user may overload an input or try to input a signal into an output terminal. The Unilab Interface is protected on all its front panel input and output connections against accidental misuse. There are, of course, limits to this protection and care must be taken to avoid exceeding these limits. Ideally, of course, the user would ensure that inputs were never overloaded.

Connection of a power supply of up to 10V AC to any front panel terminal should not cause damage. Analogue inputs can tolerate up to $\pm$ 100 volts and up to $\pm$ 35 volts can be tolerated by the analogue output. The digital inputs/outputs are the most sensitive to damage and connection of a high current source of greater than $\pm$ 10 volts for an appreciable time will result in damage to input protection resistors in the Interface. If the overload is sufficiently great to burn these out, further damage will be avoided as the input source will be disconnected from internal electronic circuitry.

7.3 SOURCES OF ANALOGUE SIGNALS

In many instances, input type transducers can be connected directly to an analogue input without need for many additional components. Whilst the actual values of the quantities you are indicating may not be easily found by this method, this is not important in many applications.

(a) Light Level

- (i) Using a photo-emissive cell
(e.g., 011.114 or 424.007)
- (ii) Using a Light Dependent Resistor
(e.g., 011.116)

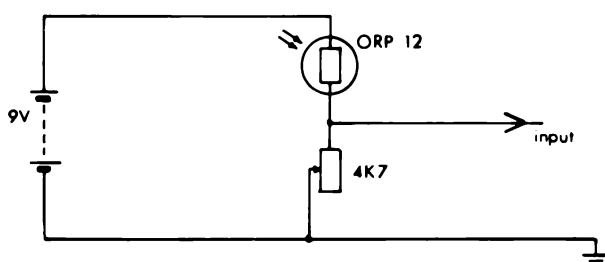


Fig. 22

(b) Temperature

Using a thermistor

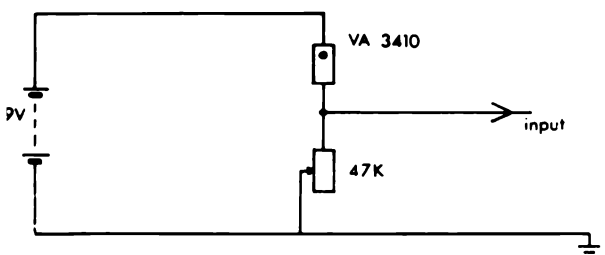


Fig. 23

(c) Sound

Using a ceramic microphone e.g., Unilab 035.141. This can be connected straight into an analogue input which is best then operated on its variable gain amplifier so that the input sensitivity can be adjusted at will. If greater sensitivity is required, the microphone output could be amplified using the amplifier in the Signal Generator/Amplifier 062.101.

(d) Connecting to Analogue Instrumentation

Many items of laboratory equipment provide output sockets for connection to external instruments such as chart recorders. These connections provide an ideal source for Interface input signals and they have the advantage that there is a known correspondence between the output from these sockets and the magnitude of the effect being measured.

A relatively low cost source of such instrumentation is provided by the Environmental Kit modules (423.001) which enable the user to measure Light Level, Oxygen Level, pH, Sound Level, Conductivity and Temperature. Up to four of these units can be connected to the Interface at any one time and the standard 0-100 mV from each unit matches the 0-100 mV input sensitivity available. Thus calibration is simple.

Other Unilab instruments which can be connected to the Interface included:

The Biological Amplifier (743.001)

an example of the use of the Interface to display ECG waveform from this amplifier is given on Page 38.

The Digital Joulemeter (082.604)

has a 0 to 1 mA output socket which provides indication of power.

The "Geigerteller" (053.002)

has a socket which provides a positive pulse for every count noted by this radioactivity meter.

Ratemeter (053.694)

can be used either with a GM tube and its power supply or the solid state detector and amplifier. By placing a resistor of 100 ohms in place of the meter link, a P.D. of 0 to 100mV across the resistor can be fed to the Interface to enable transfer of rate measurements to the computer. Alternatively, the pulse output from this unit may be used for counting purposes.

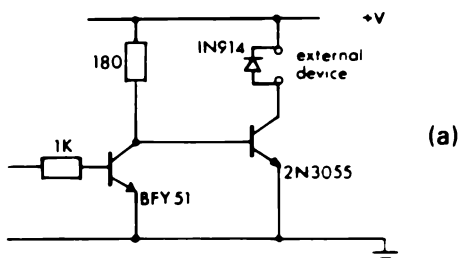
Mini Electrometer - DC amplifier (003.814)

As with the ratemeter, a 100 ohm resistor connected across the 1 mA meter sockets will provide a PD of between 0 and 100 mV for input to the Interface. This would prove a useful combination in Decay Experiments.

7.4 SOME EXTERNAL HARDWARE

1. Driving External Loads from Digital Outputs

In certain circumstances, the relays provided in the Interface may prove to be too slow in operation and the user may wish to drive an external load by means of an electronic switch. Such a situation would arise if the Interface were being used to drive a stepper motor. Circuit (a) is suitable for currents of up to about 1 amp. For higher currents, circuit (b) should be used. In both cases, the output device should be mounted on an appropriate heatsink.



NOTE this circuit inverts the output

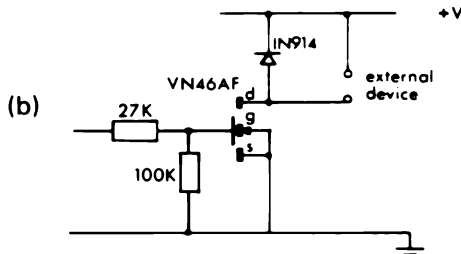


Fig. 24

2. "Debouncing" a switch input

Most mechanical switches suffer from contact bounce which can result in a number of switch closures for each switch operation. Whilst in most applications this is of little importance, contact bounce can prevent correct operation of fast electronic circuitry. Where mechanical switches are to be used with digital inputs, it is recommended that a change-over switch be used together with the following 'debounce' circuit:

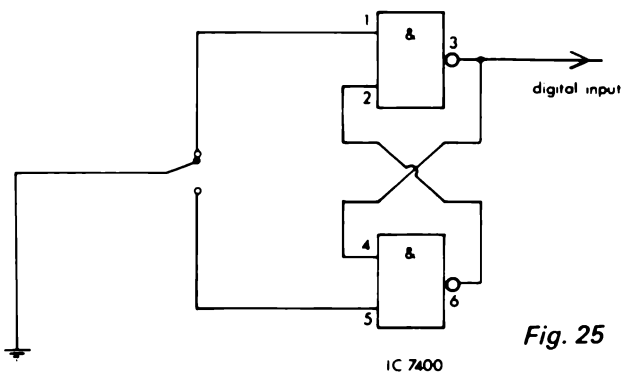


Fig. 25

3. Schmitt Input

If a digital input does not change sharply between its 0 and +5 volt levels, there will be times when the digital input circuitry will be presented with an invalid input level. To overcome this problem, a Schmitt trigger is placed between the input source and the digital input. The following circuit is suitable for this purpose:

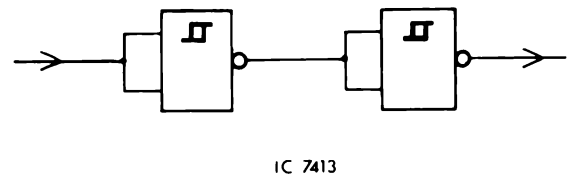


Fig. 26

4. Phototransistor Input

Often there is insufficient room on a model car to fit light-gate type phototransistors. The solution in these cases is to fit phototransistors into small collimating tubes which are then secured to the chassis of the model. Often, the phototransistor can be connected directly to the digital input on the Interface but in some instances it may be necessary to include a Schmitt trigger to provide a more decisive signal from the unit.

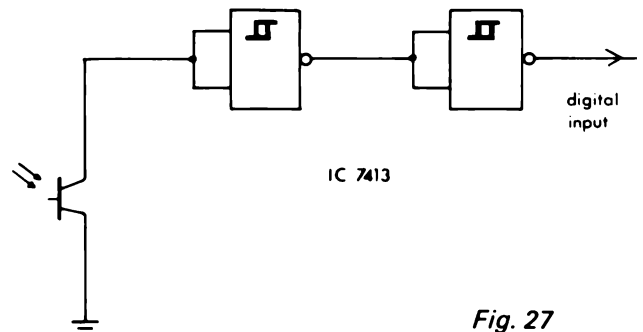


Fig. 27

7.5 THE V.I.A.

Information is passed between the computer and the Interface by means of a 6522 VIA. The large variety of possible uses of this device are far beyond the scope of this book and much has already been written about this device in the technical press.

The VIA in the Interface is located at addresses &FCC0 to &FCCF inclusive and these addresses locate the 16 registers contained in the VIA.

PORT 'A' in the VIA (&FCC1) is the port through which the computer presents the CONTROL CODE to the Interface. This port is always set as outputs on all its eight lines by placing the figure 255 in location &FCC3.

PORT 'B' in the VIA (&FCC0) is the port through which data is passed between the Interface and the computer. Depending on the mode of

operation of the Interface, the lines of this port can be set as inputs, outputs, or a mixture of inputs and outputs. The input/output status of these lines is determined by the DIRECTION CODE, a figure which is placed in location &FCC2.

Many of the programs in this book use the two timers in the VIA, T1 and T2. In general, T1 is normally used as a continuously decrementing counter in free run mode whilst T2 is often used as a counter which registers the number of times that T1 times out.

CB1, one of the VIA control lines, is used to provide the trigger facilities of the Interface.

Other facilities in the VIA, such as the shift register, are available to the user, as are the facilities of the VIA connected to the User Port in the computer itself.

7.6 BITWISE COMPARISON

In the first digital inputs programs, we looked only for changes in digital input 0. The state of a particular input selected from a number of inputs is found by performing a logical AND with the contents of all the input lines. If you look back to Page 6 you will see that line 30 started with

30 A = ?&FCC0

What this line did was to read the content of all the digital inputs and put them in A. A could then be any binary number between 00000000 (all the inputs earthed) to 11111111 (all the inputs open circuit - +5 volts). What we need to do is to "pick off" the particular input we are interested in. In the example quoted, the input we required was input 0.

Line 30 in the program continued:

B = A AND 1

What this does is to perform a logical AND between the binary contents of A and the binary number 1. For example, if all the inputs were open circuit *apart* from input 0.

A would contain binary 1 1 1 1 1 1 1 0
 while 1 in binary is 0 0 0 0 0 0 0 1
 so A AND 1 in binary is 0 0 0 0 0 0 0 0

But when input 0 becomes open circuit:

A would contain binary 1 1 1 1 1 1 1 1
 1 in binary is 0 0 0 0 0 0 0 1
 so A AND 1 in binary is 0 0 0 0 0 0 0 1

So when input 0 goes from logic 0 to logic 1, B changes from 0 to 1. This value of B is then used to decide what to do next by using IF . . . THEN statements.

If we wanted to look for a change in input line 1, then:

A might contain binary 1 1 1 1 1 1 1 1
 so we must AND with 0 0 0 0 0 0 1 0
 for B to equal 0 0 0 0 0 0 1 0

In denary, the number 00000010 is '2'.

This process is what is meant by "bitwise" comparison'.

Here is a table that tells you what numbers to use with the AND and which values are produced if an input is at logic '1':

Remember the line in the program A = ?&FCC0:B = A AND*

<i>If you want to look at input number</i>	<i>Use the following number in place of*</i>	<i>If the input is 0, B will be</i>	<i>If the input is 1 B will be</i>
0	1	0	1
1	2	0	2
2	4	0	4
3	8	0	8
4	16	0	16
5	32	0	32
6	64	0	64
7	128	0	128

Example 1.

I want to see if there is an input on input 4. From the table, * for input 4 is 16 and I will need to look for B changing from 0 to 16.

So the program on Page 6, modified to look at input 4 rather than input 0, will contain the following in lines 30, 50 and 60.

```
30 A=?&FCC0: B = A AND 16
50 IF B = 0 THEN VDU 12:PRINT "Connected to
    ground. i.e. 0 volts"
60 IF B = 16 THEN VDU 12:PRINT "Open circuit.
    i.e. +5 volts"
```

Try the program out to check that it does detect changes on input 4.

Example 2

I want to monitor two inputs, input 2 and input 5. From the table, * for input 2 is 4 and * for input 5 is 32. To monitor both we add the * values $4 + 32 = 36$.

Similarly the B values add up:

If B = 0, neither input 2 nor 5 is at logic 1
If B = 4, input 2 is at logic 1, input 5 is at logic 0
If B = 32, input 2 is at logic 0 and input 5 is at logic 1
If B = 36, both inputs 2 and 5 are at logic 1.

SECTION 8

SOFTWARE DETAILS

In general the following points apply when using the programs:

- (i) the ESCAPE key and BREAK key have been arranged to return to the start of the program.
- (ii) when expecting an input to continue, pressing the space-bar is sufficient.
- (iii) to EXIT from the program, select this option from the menu.

8.1 TIMER1

- simple timing program written in BASIC
- calculates the time in seconds for which digital input 0 is grounded
- additional equipment:
 - 1 4 mm lead
 - 1 light gate

8.2 VEL1

- simple timing program written in BASIC
- needs to be told the length of the light beam obstructor
- calculates the velocity by timing the period for which a light gate connected to digital input 0 is obstructed
- additional equipment:
 - 1 light gate
 - 1 trolley/vehicle with card of known length

8.3 VEL2

- simple timing program written in BASIC
- needs to be told the distance between two light gates
- calculates the average velocity of a vehicle travelling between light gates connected to digital inputs 0 and 1
- additional equipment:
 - 2 light gates
 - 1 trolley/vehicle with obstructor

8.4 ACCEL1

- simple timing program written in BASIC
- needs to be told the length of a light beam obstructor
- calculates the average velocity through each light gate and the effective time between the velocity measurements, hence acceleration is calculated.
- additional equipment:
 - 2 light gates
 - 1 trolley/vehicle with card of known length

8.5 ACCEL2

- simple timing program written in BASIC
- needs to be told the length of the two equal length light beam obstructors
- calculation: as with ACCEL1
- additional equipment:
 - 1 light gate
 - 1 trolley/vehicle with double obstructor of known lengths

8.6 SHRTTIM

- a timing program in BASIC with MACHINE CODE sections
- suitable for measuring times up to 130 ms with an accuracy of $\pm 15\mu\text{s}$ assuming total precision of the quartz oscillator in the computer
- timing can be started and stopped on any of the digital lines by a change from high voltage to low (+5 V to 0 V; logic 1 to logic 0) or by a change from low voltage to high (0 V to +5 V; logic 0 to logic 1): the conditions are set by a MENU but successive runs under the same conditions need not be re-selected.
- additional equipment: any system able to swing digital inputs between 0 V and +5 V, e.g. light gates, debounced mechanical switches, signal generator giving a square wave of suitable amplitude.

8.7 LONGTIM

- a timing program in BASIC with MACHINE CODE sections
- suitable for measuring times up to 4250 s with an accuracy of $\pm 60\mu\text{s}$ assuming total accuracy of the quartz oscillator in the computer
- use: as with SHORTTIMER except digital line 6 cannot be used
- additional equipment: as with SHORTTIMER

8.8 ACCEL3

- acceleration program in BASIC with MACHINE CODE timing routine
- suitable for calculating accelerations where time intervals are short
 - e.g. in measurements of 'g'
- needs to be told the length of the two equal light beam obstructors
- additional equipment:
 - 1 light gate
 - 1 double light beam obstructor

8.9 FREQCNT

- a counting program in BASIC with MACHINE CODE sections
- the program counts inputs to digital line 6 during a 'channel open' time of 65536 μ s initially, but auto-ranges to reduce this window if the count is 'over-range'
- suitable for frequencies above 2 kHz with an accuracy of $\pm 1\%$ or better
- additional equipment: any system able to swing digital input 6 between 0 and +5 V sharply, e.g., square wave from signal generator

8.10 PERIOD

- a timing program in BASIC with MACHINE CODE sections
- the program accepts inputs to analogue ports, so 'sharpening' of the signal is not necessary: high and low signal 'windows' may be set to eliminate the effect of noise
- successive 'high' and 'low' periods are timed: options then offered include calculation of frequency, 'binning' of times into 40 possible 'bins', output of results to a printer
- suitable for frequencies below 5 kHz, with an accuracy of $\pm 1\%$ or better, down to approximately 0.001 Hz
- additional equipment: system giving periodic electrical signal possibly containing noise, of amplitude between +5 V and -5 V; e.g. biological amplifier output of animal ECG

8.11 ANALOG2

- a simple analogue input program in BASIC
- a voltage less than 10 V is applied to analogue input A; the amplitude of the voltage is measured by the program after the appropriate sensitivity has been selected by the software itself; i.e. the program enables the interface to act as an auto-ranging voltmeter
- the input voltage is then displayed, together with the resolution appropriate to the sensitivity chosen by the program

8.12 GENALOG

- a program in BASIC with MACHINE CODE sections
- the full range of Interface analogue facilities is demonstrated
- by selection from a menu, one of the analogue ports is selected with appropriate sensitivity, either mono- or bi-polar: if sampling is to be started by the trigger, a rising or falling trigger voltage can be selected
- if a high resolution plot of the input is required this can be selected; 250 samples fill the screen

- the largest number of samples which can be taken is displayed: an appropriate number of samples can be selected: the period between samples can be selected from 40 μ s to 16 seconds
- once sampling has taken place, they can be plotted on a scaled grid, against time, displayed in tabular form, or output to cassette, disc, or printer
- additional equipment: system giving a time varying voltage, of amplitude at least 10 mV and less than 10 V
- note that the sampling rate chosen should be very much greater than the frequency of any periodic input, or a "strobing" effect can be created leading to inaccurate representation of the time variation of the signal.

8.13 4INPUT

- an analogue sampling program written in BASIC
- 4 analogue inputs, set for 100 mV sensitivity, monopolar, are sampled successively for approximately 60 seconds: the unscaled digital input values are displayed as sampling takes place, and then as graphs versus time; scaling can easily be added at lines 260,300,340,380 to adapt the program to the user's own probes
- the 4 channels are labelled as light, pH, temperature and oxygen level indicating the use of Unilab Environmental Probes: the program can be readily adapted to other applications
- additional equipment: e.g., 4 environmental probes and units giving 100 mV outputs e.g., Environmental Kit 423.001

8.14 FLATOUT

- an analogue sampling program written in MACHINE CODE
- a menu allows selection of input port, sensitivity and polarity: on completion, the program waits until the p.d. on the selected input port rises above the set trigger level
- the program then inputs 256 samples at the maximum possible sampling rate of 125,000 samples per second, thus allowing transients lasting less than approximately 2 ms to be examined
- the results are then displayed graphically
- additional equipment: source of transient voltage, e.g., photo-emissive cell 011.114 viewing reflected light from electronic flash unit

8.15 RLYOPT

- an analogue sampling program in BASIC with MACHINE CODE routines
- a voltage is input to port A, the variable gain having been selected by the program: the sampling rate can be selected from a menu; the longest period between samples is 100 s, the shortest period between samples is 40 μ s
- sampling is initiated by the closing of relay $\emptyset$: which can be used to switch on the voltage to be sampled.
- a second sample and display in a different colour can be taken by pressing space-bar which switches the relay off

8.16 ECG

- an analogue sampling and output program written in BASIC with MACHINE CODE routines
- a voltage to input port A is sampled at a rate selected from a menu, after a delay of 10 seconds; the variable gain amplifier and bi-polarity having been set by the program
- a graph of input p.d. against time is displayed, and can be output from the D:A port to a chart recorder, say, on pressing space-bar
- additional equipment: e.g., biological amplifier connected to give animal ECG; chart recorder with 100 mV input

8.17 TRISCAN

- an analogue sampling program written in BASIC with MACHINE CODE routines
- the voltage across the inductor in the circuit below is input at port A and displayed, after triggering by the falling voltage when lead 'X' is withdrawn from the potentiometer
- the variable capacitor can then be changed while the first plot is being drawn on the screen, the lead 'X' reinserted, sampling initiated by removing lead 'X' and the resulting waveform plotted in a different colour
- this can be performed a third time, again with a different value of C

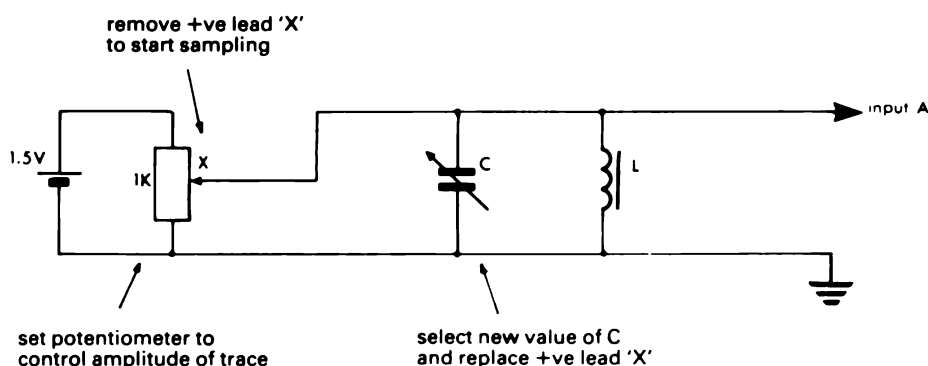
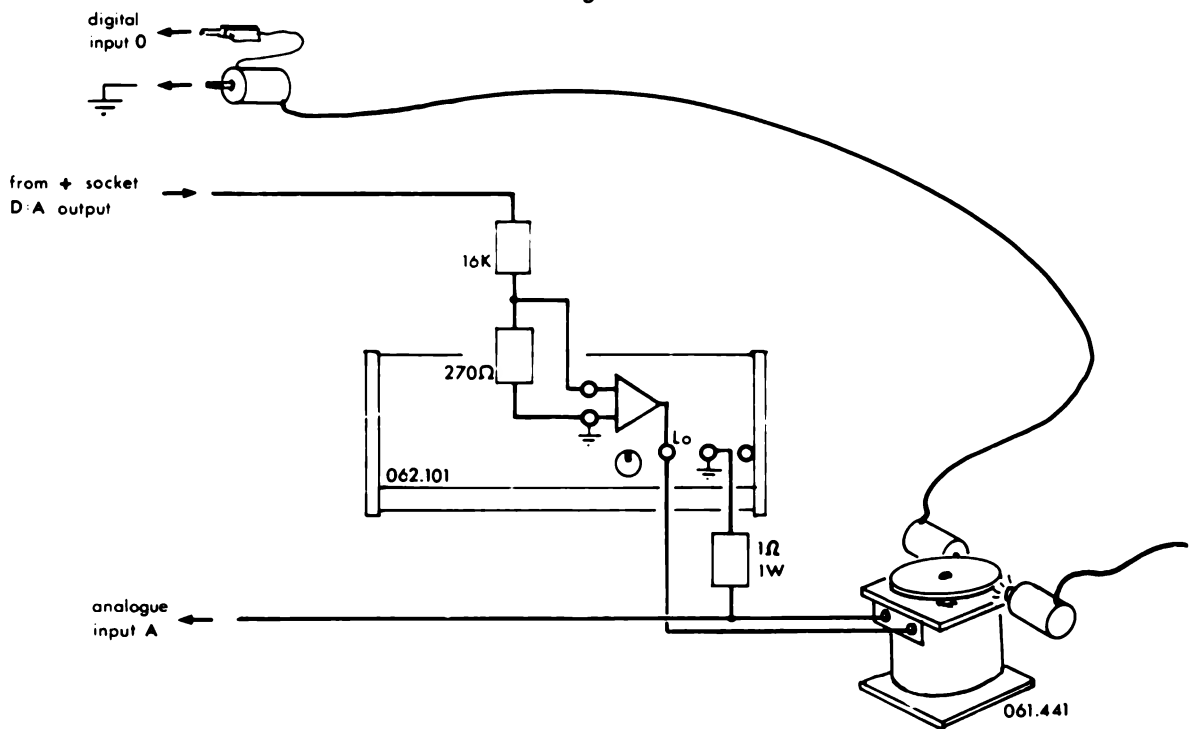


Fig. 28

8.18 BALANCE

- a program written in BASIC to simulate the system of sensing and feedback of an electronic balance
- when run, pressing the space-bar causes the program to first set the TARE position of the platform of the Vibration Generator (061.441) relative to the height of the light beam: a TARE value of 100 is suitable and can be achieved by raising or lowering the light gate connected to digital input 0.
- once the TARE position is set, a load on the platform (without obstructing the light beam) will be "ramped up" to the TARE position on pressing the space bar again; the current needed to achieve this being measured via analogue input A; this is used to calculate the weight of the load.
- note that the "ramping" current is controlled via the D:A port through a power amplifier, e.g. amplifier section of Signal Generator/Amplifier 062.101.

Fig. 29



NOTE: Automatic transfer of tape to disc

The first program on the accompanying cassette is called "T-TO-D". When loaded and run it will allow the remaining programs to be automatically loaded and then saved to disc, for those with BBC model B and discs. The copying process should take approximately fifteen minutes and is started by pressing function key 0.

For those with cassette-based systems, "GENALOG" will not operate. The equivalent cassette program is last on the tape and is entitled "GNLGCS".

Notes

Cat. No. 990.077

PRICE: £4.00

© UNILAB LIMITED 1/5/83

All Rights Reserved

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without the prior permission of the Copyright owner.



Unilab Limited
Clarendon Road, Blackburn, England BB1 9TA
Telephone (0254) 57643/4
Cables Unilab Blackburn
Telex 635775 UNILAB G